UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE UN SISTEMA DE EFECTOS ESPECIALES DE AUDIO EN TIEMPO REAL.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO ELECTRÓNICO.

P r e s e n t a:

Bogdad Roberto Carlos Espinosa Vargas.

DIRECTOR DE TESIS:
M.I. Larry Escobar Salguero

México D.F. Marzo 2008

ÍNDICE

CAPÍTULO 1.
INTRODUCCIÓN.

El sonido ha sido y será un importante medio de comunicación. Desde su nacimiento los seres humanos empiezan a desarrollar sus sentidos, pueden ver la luz del día, los colores, las imágenes, disfrutar los sabores y percibir cada sonido proveniente de su alrededor. Ya en etapas más adelante la afinidad por algún tipo de música es inevitable, que puede ir desde la música tranquila y relajante hasta la música más escandalosa.

Debido al gusto por la música y los sonidos, muy en particular la voz, se propone el siguiente trabajo donde se puede encontrar distintos efectos especiales de sonido implementados en un procesador digital de señales (DSP) en tiempo real. Dichos efectos se pueden generar naturalmente, y se pueden aplicar en la edición de sonido para alguna película o grabación musical, etc.

Existen varios artículos y trabajos acerca de los efectos de sonido, unos más completos que otros, pero la mayoría carecen del desarrollo matemático de las ecuaciones en diferencias para lograr dichos efectos, así como el análisis en el dominio temporal y de la frecuencia. Solo se concretan en presentar la función de transferencia del efecto de sonido, es decir, solo los plantean. En aquellos documentos donde si se hace una implementación lo hacen con un lenguaje muy complicado, lo que hace muy difícil reconocer los algoritmos de los efectos de sonido.

Por tal motivo, en esta Tesis se presenta de una manera sencilla el desarrollo, análisis e implementación de algunos efectos de sonido. Se explica el análisis matemático en el dominio del tiempo y de la frecuencia, auxiliado por gráficas para una mayor comprensión, y diagramas de flujo que muestran el algoritmo en un lenguaje común. Cabe mencionar que el lenguaje de programación utilizado es el lenguaje *C*, ya que es un lenguaje fácil de entender para alguien con nociones básicas de programación, es portable, existen muchos compiladores para generar código y es aplicable a muchas plataformas.

Hablando un poco más de los efectos de sonido primeramente debemos de saber que en la actualidad la mayoría de los procesos realizados sobre señales de sonido son digitales, ya que tanto el almacenamiento como el procesado y transmisión de la señal en forma digital ofrece ventajas muy significativas sobre los métodos analógicos. El procesamiento se hace en forma digital porque normalmente es más simple de realizar que el procesamiento analógico [1].

El procesamiento digital se puede llevar a cabo en varios dispositivos, por ejemplo en una computadora personal auxiliada de un software, sin embargo en esta tesis se implementará en una arquitectura DSP ya que es un dispositivo de alto desempeño y con gran futuro debido a su potencial en el procesamiento de señales.

Existe una gran cantidad de efectos musicales. Por su naturaleza podemos distinguir:

1) Efectos Basados en Retardos: La base de estos efectos es el retardo de la señal en el tiempo. Ejemplos: Eco, reverberación, regeneración, etc.

2) Efectos Basados en Filtrado: Los efectos surgen del filtrado de la señal original.

3) Efectos Basados en la Amplitud: Consisten en la alteración de la amplitud de la señal [2].

Muchos efectos se consiguen sumando a la señal original, varias copias retardadas y modificadas de diversas formas. Los más típicos son los de eco y reverberación, aunque no son los únicos. Según el efecto, los tiempos de retardos pueden variar entre las pocas milésimas y varios segundos [2].

Los efectos que se pretenden implementar en este trabajo utilizando un DSP son:

Eco. Efecto sonoro que se produce cuando un sonido rebota contra una superficie lejana y llega por duplicado al receptor con un cierto retardo. Se produce cuando las reflexiones de un sonido llegan con un retado superior a 50 ms respecto de la fuente original

Flanger. Un señal retrasada es agregada a la señal original con retraso variable (usualmente menores a 10 ms).

Reverb. Efecto que se produce al tener mas de una superficie en donde el sonido puede rebotar, lo que produce mas de un eco audible para el receptor.

Silenciar. Efecto que se obtiene al multiplicar por cero una parte o partes de una señal de audio, es decir se dejan audibles solo unas partes de la señal.

1.1 Objetivo:

Implementar un sistema capaz de procesar una señal de audio de entrada y modificarla de tal manera que a la salida presente efectos especiales de audio, perceptibles al oído humano, en tiempo real utilizando el DSP TMS320VC5402.

1.2 Definición del Problema:

Existen diversas aplicaciones, como edición de audio para cine, doblaje, edición de voz para la grabación de discos, etc., para las cuales se necesita producir ciertos efectos especiales sobre señales de audio provenientes de un micrófono o algún instrumento musical, los cuales generan una señal de audio analógica, por lo que si se quiere producir los efectos sobre esa señal directamente se necesitaría un sistema analógico capaz de producir dichos efectos. En forma analógica es muy difícil implementar efectos especiales de audio, ya que algunos de estos efectos requieren retrasar la señal o filtrarla, en cuestión del filtrado no habría mucha dificultad, sin embargo, en el proceso de retrasar la señal se pueden presentar algunas dificultades al usar componentes analógicos. Algunos de estos efectos son producidos por la naturaleza, sin embargo no son utilizables cuando son producidos de esa manera.

Lo anterior es motivo de proponer un sistema digital para realizar dichos efectos, ya que en determinados procesos y operaciones es más rápido y fácil de implementar que un sistema analógico, además de que se tendría la opción en algún momento de guardar los datos de la señal procesada o enviarlos por algún medio.

1.3 Método:

Para implementar este sistema se siguió la siguiente metodología:

- Obtener las ecuaciones matemáticas que definen a los efectos de audio para una señal discreta.
- Programar dichas ecuaciones en Matlab para observar su comportamiento y obtener un resultado ideal o modelo de cada uno de los efectos de audio.
- Trasladar dichos programas al lenguaje de programación del DSP haciendo las modificaciones necesarias para que funcionen en tiempo real y entreguen resultados similares e incluso iguales a los simulados en Matlab.
- Integrar todos los efectos de audio en un solo proyecto en el cual se tenga la opción de elegir qué efecto presente la señal de audio, y además se pueda cambiar el efecto en cualquier momento.

1.4 Resultados Esperados:

Los resultados que esperamos obtener son los siguientes:

- Funcionamiento y cambio de los efectos especiales en tiempo real.
- Sistema funcional e independiente del uso de una PC.
- Distinción entre efectos así como buena percepción de los mismos.
- Sistema funcional y atractivo en áreas relacionadas a la edición de audio y grabación musical.

1.5 Descripción de Capítulos.

En el Capítulo 2 se realiza un estudio de las señales, las señales de audio y los efectos físicos a implementar.

En el Capítulo 3 se muestran las características del DSP a utilizar y de la tarjeta de desarrollo, así como el diseño de los componentes del sistema.

El Capítulo 4 muestra la implementación final de los efectos de audio, así como la integración del sistema en el DSP.

En el Capítulo 5 se presentan los resultados de una serie de pruebas realizadas al sistema, en el dominio del tiempo y en el de la frecuencia.

El Capítulo 6 muestra las conclusiones obtenidas a partir de la implementación y funcionamiento del sistema.

Referencias.

[1] Jordá P.S. *Audio Digital y Midi, Guías Monográficas.* Anaya Multimedia. Madrid 1997.

[2] Grupo PAS. *Efectos Musicales.* Universidad de Deusto. España 2006.

CAPÍTULO 2.
SEÑALES DE AUDIO Y EFECTOS ESPECIALES

En este capítulo se puede encontrar las definiciones de señales y sus tipos, en especial la señal de audio. También se verá como se obtiene una señal digital a partir de una señal analógica, los procesos que se tienen que llevar a cabo para lograrlo, así como el Teorema de Muestreo. La parte más importante de este capítulo es la descripción de los efectos de sonido, su formación de manera natural, el análisis matemático en el dominio discreto y en el dominio de la frecuencia.

2.1 Señales.

Una señal es definida como una cantidad física que varía con el tiempo, el espacio o cualquier otra variable o variables [1]. Las señales pueden describir una variedad muy amplia de fenómenos físicos. Aunque las señales se pueden representar de muchas maneras, en todos los casos la información en una señal está contenida en un patrón de variaciones. Las señales se representan matemáticamente como funciones de una o más variables independientes [2].

Un ejemplo de una señal natural es un electrocardiograma (ECG), el cual provee al médico información acerca de la condición del corazón del paciente. Similarmente, una señal de electroencefalograma (EEG) provee información acerca de la actividad del cerebro. Las señales de ECG y EEG son ejemplos de señales que contienen información, las cuales son funciones de una sola variable independiente llamada tiempo. Un ejemplo de una señal que es función de dos variables independientes es una de imagen, en la cual las variables independientes en este caso son las coordenadas espaciales [1].

2.2 Tipos de Señales.

Existen dos tipos básicos de señales, de *tiempo continuo* y de *tiempo discreto*. En el caso de las señales de tiempo continuo la variable independiente es continua y están definidas para una sucesión continua de valores de la variable independiente, como se puede observar en la *figura 2.1*.

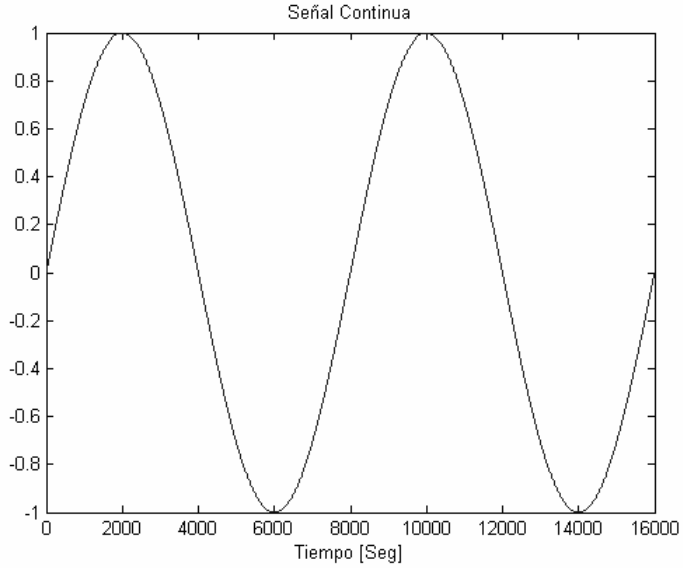

Figura 2.1 Señal Continua.

Por otra parte, las señales discretas sólo están definidas para ciertos valores de la variable independiente; en consecuencia, para estas señales la variable independiente toma sólo un conjunto de valores discretos. La *figura 2.2* representa una señal senoidal en dominio discreto. Nótese que la señal sólo toma valores cada $T/10$. Donde T es el período de la señal.

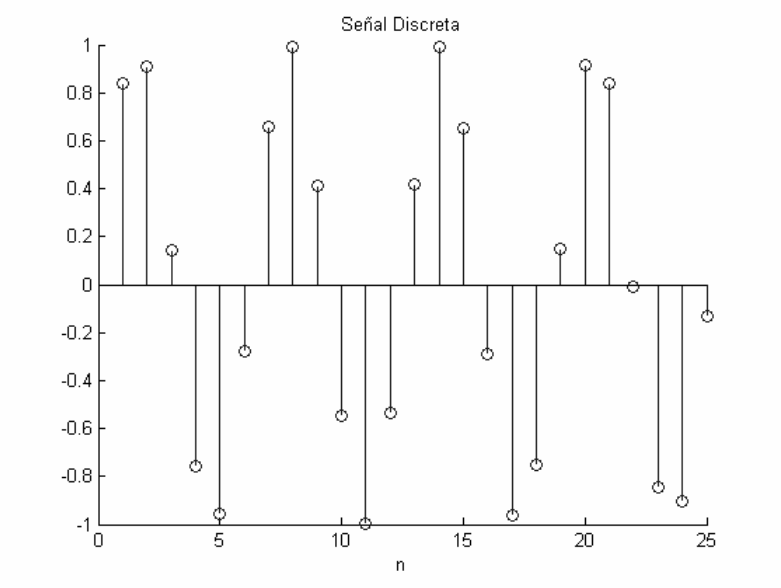

Figura 2.2 Señal Discreta.

La presión atmosférica como una función de la altitud es un ejemplo de una señal de tiempo continuo, mientras que el índice Dow Jones semanal del mercado de valores es un ejemplo de una señal de tiempo discreto [2].

2.3 Señales de Audio.

El sonido es una onda acústica que se propaga a través del aire u otros medios, formada por diferencias de presión, de forma que puede detectarse por la medida del nivel de presión en un punto. Las ondas sonoras poseen las características propias de las ondas en general, tales como reflexión, refracción y difracción [3].

El sistema vocal humano produce el habla mediante la creación de fluctuaciones en la presión acústica, los diferentes sonidos corresponden a diferentes patrones en las variaciones de la presión acústica. Por lo anterior una señal de voz se puede representar matemáticamente por la presión acústica como una función del tiempo[2], cuya gráfica se muestra en la *figura 2.3*.

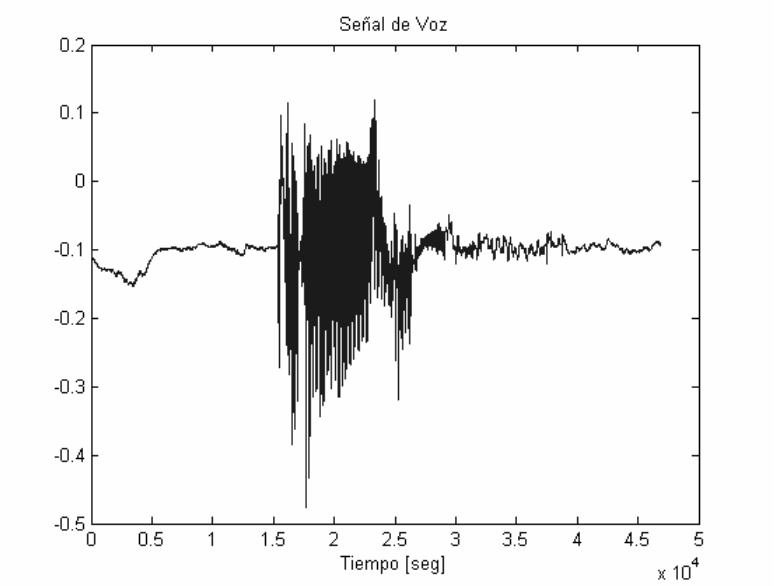

Figura 2.3 Señal de Voz

Una señal de audio se produce con variaciones de presión del aire, la podemos percibir gracias al oído que es muy sensible a las variaciones de sonido de corta duración (ms). Este proceso se lleva a cabo mediante una onda de presión que se transmite a través de un medio, como el aire y produce una sensación llamada auditiva, al perturbar el estado de reposo de las estructuras del oído. El tímpano vibra por las partículas de aire que la rodean y provoca el movimiento de los huesos del oído interno.

El oído responde al intervalo de frecuencias que podemos escuchar, el cual se conoce como el intervalo de frecuencias audibles, el cual está comprendido de manera ideal entre los 20 Hz y los 20 KHz. Aunque en la realidad el ancho de banda es más estrecho [4].

La mayoría de sonidos están compuestos por varias frecuencias diferentes. La Teoría de Fourier afirma que toda señal periódica puede descomponerse en una suma de señales sinusoidales de frecuencias y amplitudes diferentes [2].

Esta descomposición se denomina espectro de frecuencias y se representa mediante un gráfico con frecuencias en las abscisas y amplitudes en las ordenadas, en el que se visualizan las respectivas amplitudes de todas las frecuencias que componen un sonido.

Esta teoría, que data de más de un siglo, se aplica de forma rigurosa a las señales totalmente periódicas, pero los sonidos nunca lo son plenamente, pues siempre varían a lo largo del tiempo. Afortunadamente, mediante procesos matemáticos complejos, el análisis de Fourier se aplica también a señales variables en el tiempo no periódicas [5].

2.4 Procesamiento de Señales.

Algunas señales pertenecen a una clase de señales que son definidas precisamente especificando la dependencia funcional en la variable independiente. Sin embargo, existen casos en los que dicha relación funcional es desconocida o muy complicada para un uso práctico.

Por ejemplo, la voz no puede ser descrita funcionalmente por ninguna expresión matemática sencilla. Algunas señales pueden ser representadas con un alto grado de precisión como la suma de varias senoidales con diferentes amplitudes y frecuencias esto es:

$$\sum_{i=1}^{N} A_i(t)sen[2\pi F_i(t)t + \theta_i(t)] \qquad (2.1)$$

Donde $\{A_i(t)\}$, $\{F_i(t)\}$, y $\{\theta_i(t)\}$ son los conjuntos de (posiblemente variables con el tiempo) amplitudes, frecuencias, y fases, respectivamente de las senoidales. De hecho, una forma de interpretar la información o el mensaje contenidos en cualquier segmento pequeño de una señal de voz, es medir las amplitudes, frecuencias y fases contenidas en el segmento de la señal.

Asociado con las señales naturales están los medios por los cuales dichas señales son generadas. Por ejemplo, las imágenes son obtenidas al exponer una película fotográfica a una escena o un objeto. Por eso la generación de señales está usualmente asociada con un sistema que responde a un estimulo o fuerza. En una señal de voz, el sistema consiste de las cuerdas vocales y el tracto vocal, también llamado cavidad vocal. El estimulo en combinación con el sistema es llamado *origen* de la señal. Entonces tenemos orígenes de voz, orígenes de imágenes, y otros varios tipos de orígenes de señales.

Un sistema también puede ser definido como un dispositivo físico que realiza una operación sobre una señal. Cuando pasamos una señal a través de un sistema decimos que hemos procesado la señal, en general, el sistema se caracteriza por el tipo de operación que realiza sobre la señal. Por ejemplo, si la operación es lineal, el sistema es llamado lineal. Si la operación sobre la señal es no lineal, el sistema es llamado no lineal, y así sucesivamente. Dichas operaciones son usualmente referidas como *procesamiento de señales.*

Para el propósito de este trabajo, es conveniente aclarar que la definición de sistema no incluye solo dispositivos físicos, sino también a las operaciones realizadas a través de un software sobre una señal. En procesamiento digital de señales en una computadora digital, las operaciones realizadas sobre una señal consisten en un número de operaciones matemáticas especificadas en un software que realiza dicho proceso [1].

2.4.1 Elementos Básicos de un Sistema de Procesamiento Digital de Señales.

Muchas de las señales encontradas en ciencia e ingeniería son analógicas por naturaleza. Dichas señales pueden ser procesadas directamente por sistemas analógicos apropiados con el propósito de cambiar sus características o extraer alguna información deseada. En ese caso se dice que la señal ha sido procesada directamente en su forma analógica. Ambas señales (entrada y salida)

están en forma analógica [1]. La *figura 2.4* presenta el diagrama de bloques básico del Procesamiento Analógico de Señales.

Figura 2.4 Procesamiento Analógico de Señales.

El Procesamiento Digital de Señales provee un método alternativo para procesar señales analógicas, involucrando una interfaz entre la señal analógica y el procesador digital. Dicha interfaz es conocida como conversor analógico-digital (ADC por sus siglas en Inglés), la salida del ADC es una señal digital, que es una entrada apropiada para el procesador digital de señales [1]. En la *figura 2.5* se puede observar el diagrama de bloques básico de un sistema de Procesamiento Digital de Señales.

Cabe señalar que en la entrada y/o en la salida del procesador digital, se pueden agregar señales que previamente han sido adecuadas o que por su naturaleza son digitales.

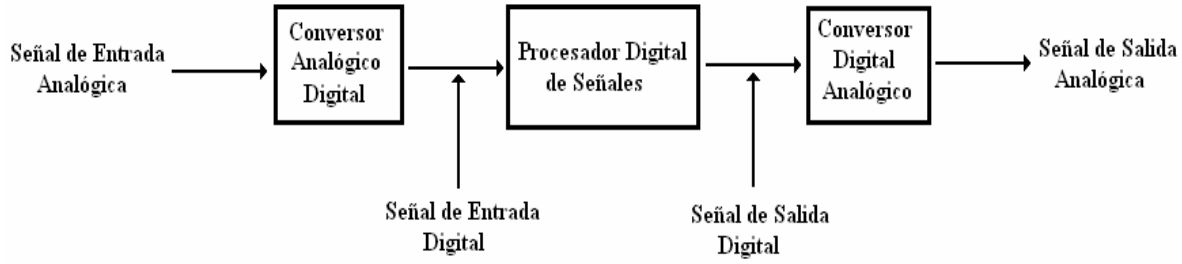

Figura 2.5 Procesamiento Digital de Señales.

2.4.2 Conversión Analógica Digital.

Para procesar señales analógicas por medios digitales es necesario primero transformarlas a una forma digital; esto es, transformarlas en una secuencia de números teniendo una precisión finita, este proceso es llamado conversión analógica digital (A/D).

El proceso de transformación consta de tres pasos:

- Muestreo: Consiste en obtener "muestras" de la señal de tiempo continuo en instantes de tiempo discretos. Esto es, si $x_a(t)$ es la entrada del muestreador, la salida es $x_a(nT) \equiv x(n)$, donde T es llamado tiempo o intervalo de muestreo.
- Cuantización: Es trasladar una señal de tiempo discreto y valor continuo a una señal de tiempo discreto y valor discreto (digital). El valor de cada muestra de la señal es representado por un valor seleccionado entre un conjunto finito de posibles valores. La diferencia entre la muestra sin cuantizar $x(n)$ y la salida cuantizada $x_q(n)$ es llamado error de cuantización.

- Codificación: En este proceso, cada valor discreto $x_q(n)$ es representado por una secuencia binaria de n bits [1].

En la *figura 2.6* se muestra una señal senoidal y su aproximación de acuerdo al número de bits establecidos en el proceso de codificación.

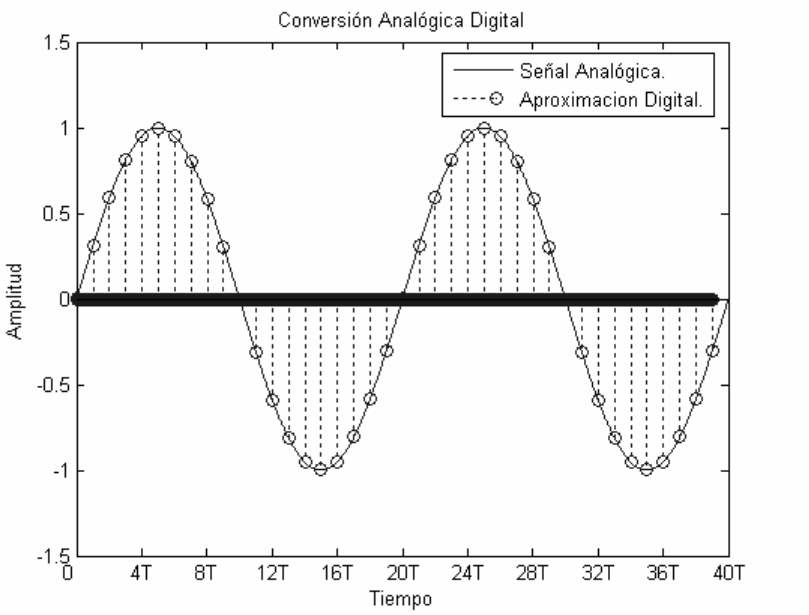

Figura 2.6 Conversión Analógica Digital.

2.4.2.1 Muestreo de Señales Analógicas.

Existen muchas maneras para muestrear una señal analógica, la más común y usada es el muestreo uniforme descrito por:

$$x(n) = x_a(nT) \quad -\infty < n < \infty \tag{2.2}$$

Donde $x(n)$ es la señal de tiempo discreto obtenida al tomar muestras de la señal analógica $x_a(t)$ cada T segundos. El reciproco $1/T = F_S$ es llamado *tasa de muestreo* (muestras por segundo) o *frecuencia de muestreo* (Hz).

El muestreo periódico establece una relación entre las variables de tiempo t y n de las señales de tiempo continuo y tiempo discreto respectivamente. Estas variables están relacionadas a través del período de muestreo T o bien, a través de la tasa de muestreo $F_S = 1/T$, como sigue: [1]

$$t = nT = \frac{n}{F_S} \tag{2.3}$$

2.4.2.2 Teorema de Muestreo.

Dada una señal analógica, debemos seleccionar el período de muestreo T o, equivalentemente, la frecuencia de muestreo F_S, para lo cual debemos tener alguna información acerca de las características de la señal a ser muestreada. Generalmente sabemos que la mayor parte de los componentes de frecuencia de una señal de voz caen debajo de los 3500 Hz.

Sabemos que la frecuencia más grande en una señal analógica que puede ser reconstruida sin ambigüedad cuando la señal es muestreada a una tasa de $F_S = 1/T$ es $F_S/2$. Cualquier frecuencia encima de $F_S/2$ resultan muestras que son idénticas con la correspondiente frecuencia en el intervalo $-F_S/2 \le F \le F_S/2$. Para evitar ambigüedades resultantes por el aliasing, debemos seleccionar una tasa de muestreo lo suficientemente grande. Esto es, debemos seleccionar $F_S/2$ mayor que $F_{\max}$ entonces:

$$F_S > 2F_{\max} \tag{2.4}$$

La teoría de comunicaciones establece que si la frecuencia más grande contenida en una señal analógica $x_a(t)$ es $F_{\max} = B$ y la señal es muestreada a una tasa de $F_S > 2F_{\max} \equiv 2B$, entonces $x_a(t)$ puede ser exactamente recuperada de sus valores muestreados usando la función de interpolación:

$$g(t) = \frac{sen(2\pi Bt)}{2\pi Bt} \tag{2.5}$$

Entonces $x_a(t)$ se puede expresar como:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{F_S}\right) g\left(t - \frac{n}{F_S}\right) \tag{2.6}$$

Donde $x_a\left(n/F_S\right) = x_a(nT) \equiv x(n)$ son las muestras de $x_a(t)$ [1].

2.4.3 Ventajas del Procesamiento Digital de Señales sobre el Analógico.

Existen muchas razones por las cuales el procesamiento digital de una señal analógica es preferible al analógico. Entre ellas tenemos el hecho de que un sistema digital permite flexibilidad, para reconfigurar las operaciones del procesamiento digital de señales simplemente cambiando el programa. La reconfiguración de un sistema analógico implica un rediseño del hardware seguido de pruebas y verificación para observar si este opera apropiadamente.

Las consideraciones de precisión también juegan un papel importante en la determinación de la forma del procesador de señales. Las tolerancias en los componentes analógicos de los circuitos

presentan una dificultad mayor al diseñador del sistema, ya que el control de estas tolerancias es mínimo y provocan un intervalo mayor de inestabilidad. Por otro lado, un sistema digital proporciona un mejor control en los requerimientos de precisión. Dichos requerimientos resultan en especificar los requerimientos de precisión en el ADC y el procesador digital de señales, en términos de longitud de palabra, aritmética de punto flotante o de punto fijo, y factores similares.

En muchos casos la implementación digital del sistema de procesamiento de señales es más económica que su contraparte analógica. Como consecuencia de estas ventajas, el procesamiento digital de señales ha sido aplicado en sistemas prácticos cubriendo un gran intervalo de disciplinas, por ejemplo su aplicación en procesamiento de voz y transmisión de señales en canales telefónicos, en procesamiento y transmisión de imágenes, sismología, geofísica, exploración petrolera, detección de explosiones nucleares, procesamiento de señales recibidas desde el espacio y en una vasta variedad de otras aplicaciones [1].

2.5 Efectos Especiales de Audio.

Muchos sistemas de grabación, amplificación o generación de señales de audio tienen una parte dedicada a los efectos, incluso hay módulos especializados para la generación de efectos. Un efecto para una señal de audio es, a grandes rasgos, la modificación de una señal de entrada para conseguir que el sonido alcance características en concreto a la salida [6].

Existen gran cantidad de efectos de Audio. Por su naturaleza podemos distinguir [7]:

- Efectos basados en retardos.
- Efectos basados en filtrado.
- Efectos basados en la amplitud.

2.5.1. Efectos Basados en Retardos.

Muchos efectos se consiguen sumando a la señal original copias retardadas y modificadas de diversas formas, siendo los más típicos los de eco y reverberación, aunque no son los únicos. Según el efecto, los tiempos de retardos pueden variar entre pocas milésimas y varios segundos [5].

2.5.1.1 Eco.

Es un efecto sonoro que se produce cuando un sonido rebota contra una superficie lejana y llega por duplicado al receptor con un cierto retardo. Se produce cuando las reflexiones de un sonido llegan con un retado superior a 50 ms respecto de la fuente original. En forma analógica este efecto se obtenía gracias a las dos cabezas magnéticas (grabación y reproducción). Introduciendo un sonido, grabándolo y reproduciéndolo inmediatamente producía un retardo cuyo tiempo estaba determinado por la distancia entre las cabezas y por la velocidad de la cinta. Actualmente se consiguen mediante retardos digitales (delays) que nos permiten tiempos desde una milésima de segundo hasta 3 ó 4 segundos [7].

Un efecto de eco básico puede ser obtenido simplemente sumando una muestra de sonido anterior a la muestra de sonido actual [8]. En la *figura 2.7* se puede apreciar el modelo físico de una señal eco.

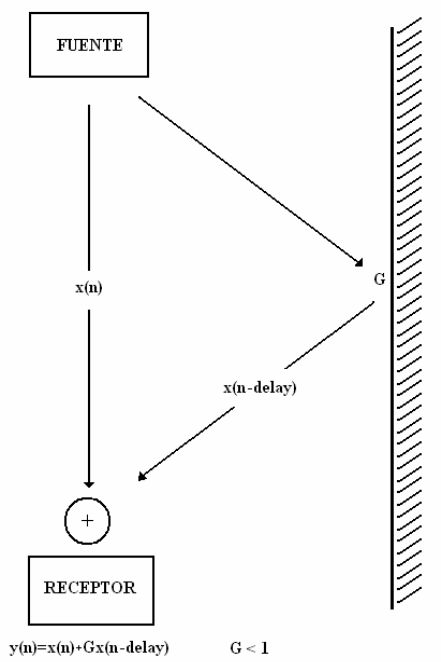

Figura 2.7 Formación de Eco en la naturaleza.

La señal original se representa como $x(n)$ y la señal con retardo como $x(n-delay)$, donde $delay$ es el retardo, G la ganancia y $y(n)$ la señal de resultante.

En el dominio discreto se tiene:

$$y(n) = x(n) + Gx(n-d) \tag{2.7}$$

Pasando al dominio de la frecuencia mediante la transformada Z se obtiene:

$$Y(z) = X(z) + Gz^{-d}X(z) \tag{2.8}$$

De la *ecuación (2.8)*, despejando $X(z)$ se obtiene la función de transferencia del sistema.

$$\frac{Y(z)}{X(z)} = 1 + Gz^{-d};\ G < 1 \tag{2.9}$$

2.5.1.2 Reverberación (Reverb).

La Reverberación se obtiene con la suma total de las reflexiones del sonido que llegan al lugar del oyente en diferentes momentos del tiempo. En una sala, la reverberación se produce de forma natural porque los sonidos que nos llegan a los oídos no proceden de un único punto emisor, sino que también recibimos "copias" reflejadas por las paredes, el techo, el suelo y otros objetos [7].

En términos prácticos el efecto Reverberación se puede definir como una suma de efectos Eco, con diferentes ganancias que obviamente son menores a 1. En la *figura 2.8* se puede observar

como al receptor llegan diferentes señales como son $x(n)$, $x(n-delay_1)$ y $x(n-delay_2)$. Cabe mencionar que en este efecto se pueden considerar un número infinito de señales es decir $x(n-delay_m)$ donde $m \to \infty$.

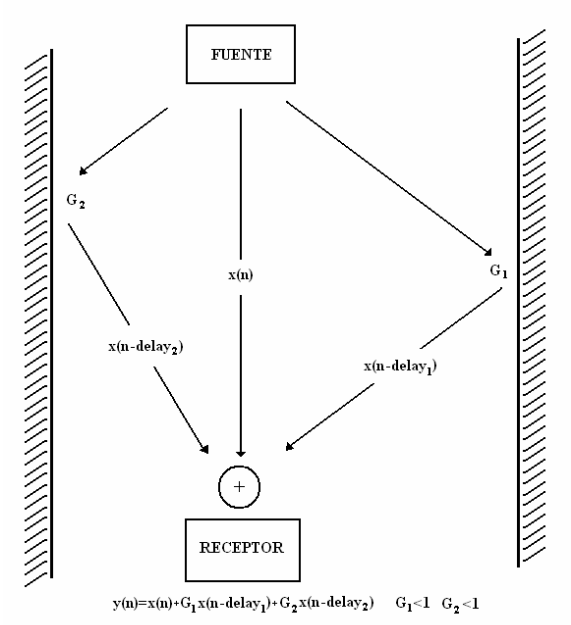

Figura 2.8 Formación de Reverb en la naturaleza.

Por lo tanto, la señal obtenida por reverberación, en el dominio discreto será:

$$y(n) = x(n) + G_1 x(n-d_1) + G_2 x(n-d_2) + ... + G_m x(n-d_m) \tag{2.10}$$

Pasando al dominio de la frecuencia mediante la transformada Z se obtiene:

$$Y(z) = X(z) + G_1 z^{-d_1} X(z) + G_2 z^{-d_2} X(z) + ... + G_m z^{-d_m} X(z) \tag{2.11}$$

De la ecuación 2.11, despejando X(z) se obtiene la función de transferencia del sistema.

$$\frac{Y(z)}{X(z)} = 1 + G_1 z^{-d_1} + G_2 z^{-d_2} + ... G_m z^{-d_m};\ G_1 < 1\ y\ G_2 < 1 \tag{2.12}$$

2.5.1.3 Flanger.

Es un efecto, con base al retardo, usado en los estudios de grabación desde 1960, que se obtiene sumando las salidas de dos maquinas grabadoras reproduciendo la misma pista mientras se toca la orilla de uno de los carretes para frenarlo, creando un retraso entre las dos maquinas. La orilla del primer carrete se va liberando poco a poco mientras se toca la orilla del otro carrete, causando que el retraso desaparezca gradualmente y después crezca en la dirección opuesta. En este caso no se escucha eco debido a que los retrasos son muy cortos, normalmente entre 1 y 10 ms [8].

2.5.2 Efectos Basados en Filtrado.

Estos efectos se producen al introducir la señal a procesar a través de un sistema que cambie sus propiedades en el dominio de la frecuencia (espectro), o bien sumándole a la señal original una muestra modificada espectralmente. El efecto más conocido que utiliza filtros es la ecualización.

Un filtro se caracteriza por su curva de respuesta en frecuencia, que indica la forma en que las diferentes frecuencias en la entrada se atenúan o amplifican. Todos los componentes electrónicos de sonido poseen una curva de respuesta de frecuencia particular, aunque en la mayoría de aparatos lo ideal sería que esta curva fuese plana entre los 20 Hz y los 20 kHz, ya que toda desviación acarrea una modificación artificial del timbre.

Los filtros más usuales pueden clasificarse, de acuerdo con la forma de esta curva de respuesta, en cuatro grandes familias: paso bajas, paso altas, paso banda y supresor de banda.

- El filtro Paso Bajas, deja pasar las frecuencias por debajo de un determinado valor, denominado *frecuencia de corte (f_c)*. En un filtro ideal esta frecuencia debería suponer una discontinuidad en la curva de respuesta, de forma que toda frecuencia por encima de este valor se atenuará totalmente y toda frecuencia por debajo se dejará tal cual. En la práctica, esto no es posible y todos los filtros reales presentan una pendiente en la zona cercana a la frecuencia de corte. Cuanto más inclinada sea esta pendiente de atenuación, el filtro será de más calidad. En la *figura 2.9* se muestra la gráfica de un filtro paso banda ideal, donde f = frecuencia, $M(f)$ = Magnitud y f_c = frecuencia de corte.

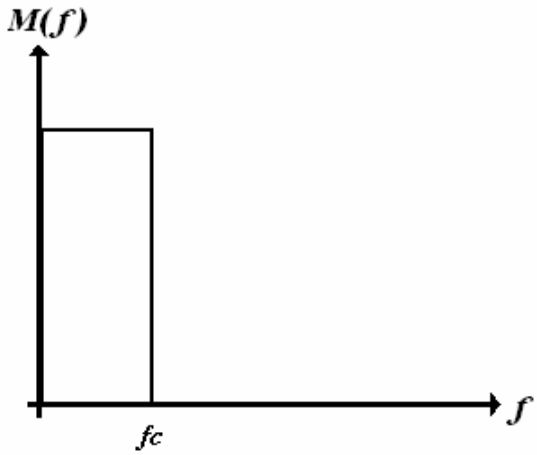

Figura 2.9 Filtro Paso Bajas Ideal

- El Filtro Paso Altas, realiza la labor opuesta, ya que únicamente deja pasar las frecuencias superiores a la frecuencia de corte, como se muestra en la *figura 2.10.*

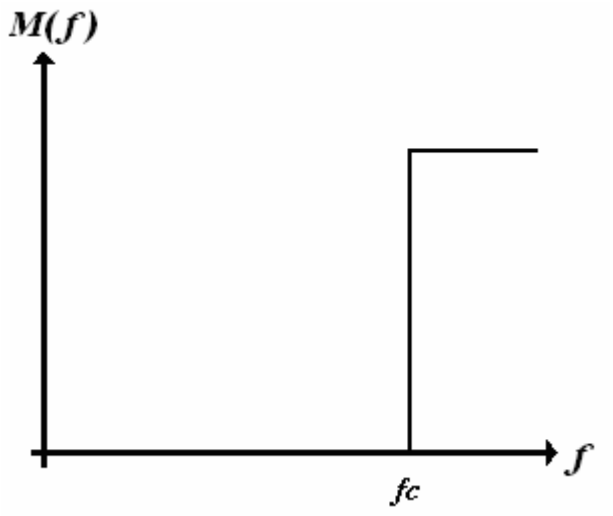

Figura 2.10 Filtro Paso Altas Ideal

- El Filtro Paso Banda deja pasar una banda de frecuencias, eliminando el resto, como se muestra en la *figura 2.11*. Se puede definir de dos maneras, ya sea a partir de la frecuencia central y su ancho de banda, o bien a partir de las frecuencias f_1 y f_2. Donde la relación entre la frecuencia de corte y las frecuencias f_1 y f_2 está dada por:

$$fc = \frac{f_1 + f_2}{2} \tag{2.13}$$

Y para el caso del ancho de banda se tiene:

$$BW = f_2 - f_1 \tag{2.14}$$

Donde BW = Ancho de Banda (Band Width) por sus siglas en Inglés.

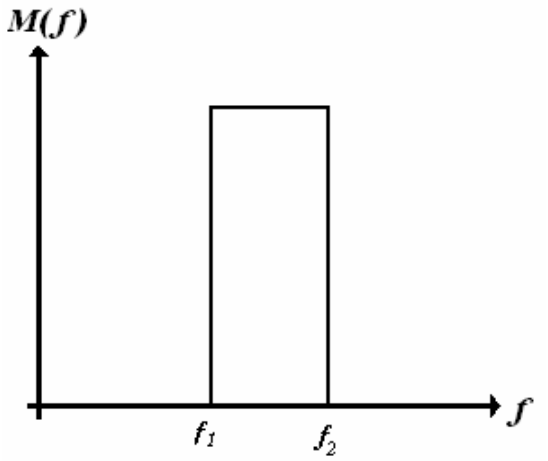

Figura 2.11 Filtro Paso Banda Ideal

- El Filtro Supresor de Banda, actúa de forma inversa al paso banda. Al igual que éste, se caracteriza por la frecuencia de corte y el ancho de banda o por las frecuencias f_1 y f_2. [5], como se muestra en la *figura 2.12.*

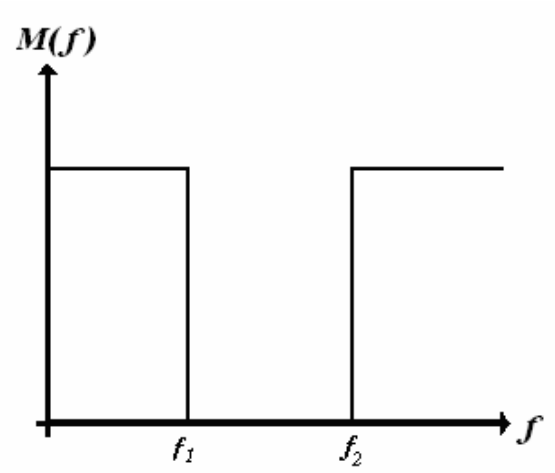

Figura 2.12 Filtro Supresor de Banda Ideal

2.5.2.1 Ecualización.

Se realiza mediante la implementación de un banco de filtros paso banda en paralelo, el cual debe tener control sobre la ganancia de cada filtro de manera individual y la salida de todos los filtros debe ser sumada para obtener la señal de salida. El proceso de ecualización se refiere a la modificación en la ganancia de las bandas de los filtros en la señal de entrada, si dicha ganancia es igual en todos los filtros y es máxima, la señal de salida se escuchará idéntica a la señal de entrada, pero si se reduce la ganancia de uno o varios filtros, la señal de salida se escuchará diferente.

2.5.3 Efectos Basados en la Amplitud.

Estos efectos se logran manipulando la amplitud o la ganancia de la señal de salida de audio. El efecto más conocido y sencillo de este tipo es la amplificación, ya que solo se multiplica la señal de entrada por una cierta ganancia. Sin embargo, existen efectos más complicados en su elaboración que modifican la amplitud.

2.5.3.1 Trémolo.

Consiste en una modulación de amplitud (AM) de la señal de entrada. La señal portadora es la señal de audio, mientras que la señal moduladora es típicamente una señal senoidal de baja frecuencia (normalmente entre 1 Hz y 20 Hz). Matemáticamente, el Trémolo se expresa mediante la fórmula de modulación AM.

$$S_{AM}(t) = A[1 + am_n(t)]\cos(\omega t) \qquad (2.15)$$

Donde a es el índice de modulación (comprendido entre 0 y 1) y ω es la frecuencia de la señal moduladora [6].

2.5.3.2 Cambio de Pitch.

El pitch puede ser definido como la frecuencia fundamental de una señal de audio, en especial una de voz. Debido a que las señales de voz no son periódicas y a la vez son aleatorias, y que por tanto no se pueden representar mediante una ecuación matemática, no existe una manera de conocer su frecuencia exacta, ya que es la mezcla de varias frecuencias, sin embargo se conoce el ancho de banda en el que se encuentra, y mediante el análisis de Fourier se conoce su frecuencia fundamental o pitch, alrededor de la cual se encuentran las demás frecuencias que componen la voz.

Una manera sencilla de cambiar el pitch de una señal de voz se realiza mediante varias modulaciones en AM de la señal y procesos de filtrado, lo cual desplaza el espectro de la señal, provocando un cambio en la frecuencia fundamental de la voz dando como resultado una señal de voz ya sea mas grave o mas aguda dependiendo del sentido de corrimiento en frecuencia.

2.5.3.3 Vibrato.

Es un efecto muy usado en estudios de grabación, ya que mediante este efecto se consigue que la voz suene afinada. La ecuación básica del vibrato es:

$$V_S(t) = m(t) + sen(2\pi f_V) \tag{2.16}$$

Donde $m(t)$ es la señal original y f_V es la frecuencia del vibrato.

2.5.3.4 Otros.

Existe otra variedad de efectos de audio, tales como:

Puerta de Ruido: Silencia las muestras por debajo de determinado umbral, introducido como parámetro.

Silenciar: Multiplica por 0 la zona seleccionada y da una amplitud al resto de la señal.

Invertir: Se realiza una reflexión de la señal respecto al eje horizontal. Los valores positivos pasan a ser negativos y viceversa. Se percibe mejor cuando se aplica a un canal único de un sonido estéreo [7].

2.6 Resumen.

- Las señales de audio tienen un intervalo de frecuencias que va desde los 20 Hz hasta los 20 KHz idealmente, ya que cada persona tiene un ancho de banda auditivo menor.
- Existen dos tipos de procesamiento de señales, el procesamiento analógico y el digital, teniendo este último mayores ventajas.
- La conversión analógica – digital consta de tres procesos: muestreo, cuantización y codificación
- El Teorema de muestreo nos dice que para evitar ambigüedades resultantes por el aliasing, debemos seleccionar una tasa de muestreo mayor que dos veces la frecuencia máxima de la señal analógica.
- Los efectos especiales de audio se pueden clasificar en tres: los basados en retardos, en filtrado y en amplitud.
- El eco, la reverberación y el flanger son ejemplos de efectos basados en retardos; los cuatro tipos de filtros (paso bajas, paso altas, paso banda y supresor de banda) así como la ecualización son efectos basados en filtrado; el trémolo, cambio de pitch y vibrato, son ejemplos de efectos basados en la amplitud.

Referencias

[1] Proakis J.G. *Digital Signal Processing, Principles, Algorithms and Applications*. Prentice Hall. Tercera Edición. E.E.U.U. 1996.

[2] Oppenheim A.V. *Señales y Sistemas.* Prentice Hall Hispanoamericana S.A. México 1994.

[3] http://www.hispamp3.com/tallermp3/tutoriales/mp3profundidad/2.shtml

[4] Jordá P.S. *Audio Digital y Midi, Guías Monográficas.* Anaya Multimedia. Madrid 1997.

[5] Guaus E. *Implementación de Efectos para Señales de Audio.* Departamento de Acústica Universidad Ramon Llull. Barcelona, España.1996.

[6] Grupo PAS. *Efectos Musicales.* Universidad de Deusto. España 2006.

[7] Micea M.V. *Implementing Professional Audio Effects with DSPs.* Software and Computer Engineering Department, Politechnica University of Timisoara, Rumania.1998.

[8] Smyth T. *CMPT 889: Lecture 7 Delay Effects: Flanging, Phasing, Chorus, Artificial Reverb.* School of Computing Science, Simon Fraser University. 2000.

CAPÍTULO 3.
DISEÑO Y DESARROLLO DEL SISTEMA

El diseño y desarrollo del sistema de efectos especiales de audio en tiempo real requiere un análisis tanto de los efectos a realizar, como de los elementos físicos necesarios para su correcto funcionamiento, esto es, el dispositivo empleado para introducir la señal de audio al sistema, el dispositivo empleado para escuchar la señal de salida, los componentes internos de la tarjeta de desarrollo y el DSP a utilizar.

En relación con los efectos de audio es importante diseñar los algoritmos necesarios para su implementación, primero para un funcionamiento correcto y que posteriormente sea posible modificarlos para su uso en tiempo real.

En este capítulo se analizará el hardware del sistema, es decir, el micrófono, el convertidor analógico digital, el amplificador, la bocina y desde luego el DSP. De ese análisis se procederá al diseño de los componentes del sistema, como son los efectos de Eco, Reverb, Flanger y Silenciar.

3.1 Hardware del Sistema.

Una señal de audio es la representación eléctrica de una onda sonora, normalmente acotada al intervalo de frecuencias audibles por los seres humanos, entre los 20 Hz y los 20 kHz, aproximadamente. Dado que el sonido es una onda de presión, se requiere de un transductor de presión (un micrófono) que convierte las ondas de presión de aire en señales eléctricas. La conversión contraria se realiza mediante una bocina, que convierte las señales eléctricas en ondas de presión de aire.

Una señal de audio se puede caracterizar por su intervalo dinámico, potencia, relación señal-ruido, o por su espectro de potencia, ancho de banda, frecuencia fundamental, armónicos, distorsión armónica, etc. Dicha caracterización se utiliza para llevar a cabo la conversión de la señal analógica a señal digital, ya que el convertidor analógico digital (ADC) recibe los valores de la señal de entrada analógica para convertirlos a valores digitales; es decir, a los valores de voltaje de entrada, les asocia un valor binario. Un ejemplo de este proceso se puede observar en la *tabla 3.1*.

Entrada	Salida
0.00 [V]	00000000
0.02 [V]	00000001
0.04 [V]	00000010
0.06 [V]	00000011
0.08 [V]	00000100
…	…
1 [V]	00110011
…	…
4.98 [V]	01111110
5.00 [V]	01111111

Tabla 3.1 Relación de Volts a Número Binario.

Un sistema se puede definir como el conjunto de procesos individuales que trabajan para realizar una tarea. El Sistema de Procesamiento de señales de Audio implementado consta de varias etapas, las cuales se pueden apreciar en la *figura 3.1* y se describe a continuación.

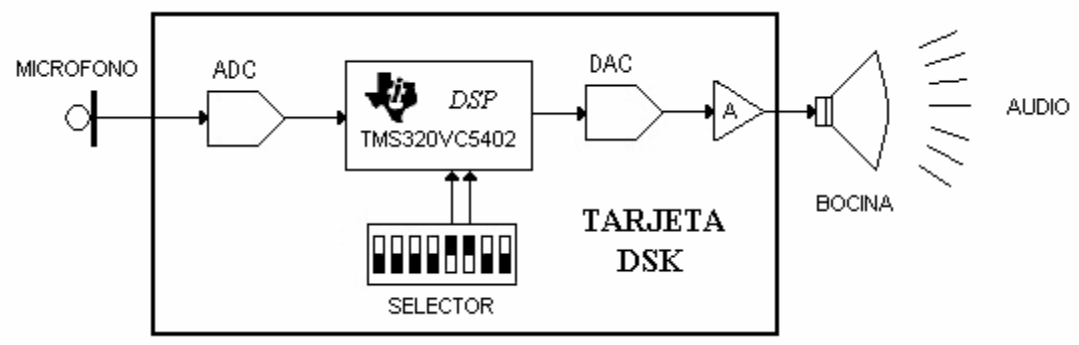

Figura 3.1 Diagrama General del Sistema.

3.1.1 Micrófono.

Para conectar la entrada y salida de audio, es decir el micrófono y la bocina respectivamente, a la tarjeta de desarrollo a utilizar (DSP estándar kit DSK), se tienen dos conectores estándares de 3.5 mm, un conector para la entrada de audio desde un micrófono y el otro para conectar la salida de audio a una bocina. La entrada de audio está diseñada para acoplar señales analógicas e incluye un amplificador de hasta 10 dB de ganancia y una etapa de conversión que permite que la señal pase de un estado "*single-ended*" (o de una sola terminación) a un estado diferencial, antes de ser digitalizada por el codec TLC320AD50C.

La entrada de audio está diseñada para micrófonos electret, ya que cuenta con una polarización interna, por tanto, si se quiere conectar a la entrada un micrófono dinámico se necesitará desacoplar el voltaje de polarización, es por esto que el micrófono que se usa para la implementación de este sistema, es un micrófono tipo electret. El nivel de voltaje de entrada permitido es de 500 mV (350 mVrms), este nivel se puede amplificar en la tarjeta DSK debido a que tiene la posibilidad de programar ganancias de 0 a 12 dB en incrementos de 6 dB mediante software de control. Para el caso de la salida de audio también se puede configurar mediante software para elegir las ganancias de salida desde 0 a -12 dB con decrementos de 6 dB. Cabe mencionar que la configuración en hardware da la interfaz de salida soporta cargas que pueden ir de 8 Ω hasta 600 Ω [1].

El funcionamiento del micrófono electret se basa en el concepto que relaciona el voltaje (V), la capacitancia (C) y la carga (Q), esta relación se observa en la *ecuación (3.1)*:

$$V(t) = \frac{Q(t)}{C(t)} \tag{3.1}$$

Cuando existe una señal sonora en el micrófono, se produce un movimiento en el par de placas del capacitor, que la estar cargadas, inducen el movimiento de los electrones, produciendo pulsos eléctricos que son codificados por el convertidor analógico-digital.

Los micrófonos electret tienen una respuesta en frecuencia de 50 a 15,000 Hz. Un micrófono electret empieza a indicar que debe ser retirado (que ha acabado su vida activa) cuando produce zumbidos o ruidos inexplicables.

3.1.2 Convertidor Analógico-Digital, Digital-Analógico.

Para lograr la conversión analógica – digital y digital – analógica, la tarjeta DSK consta de un codec con varias etapas. Para la entrada estas etapas son: la etapa de pre-amplificación, filtrado antialiasing y conversión diferencial para la entrada del convertidor. Para la salida se tiene una etapa de conversión unitaria y un filtro paso bajas [2]. La interconexión de estas etapas se puede apreciar en la *figura 3.2.*

El convertidor tiene dos canales para realizar la operación sobre señales estéreo. Cabe mencionar que los sistemas estéreo tienen dos señales de salida, mientras que los sistemas mono como su nombre lo indica, solo tienen una.

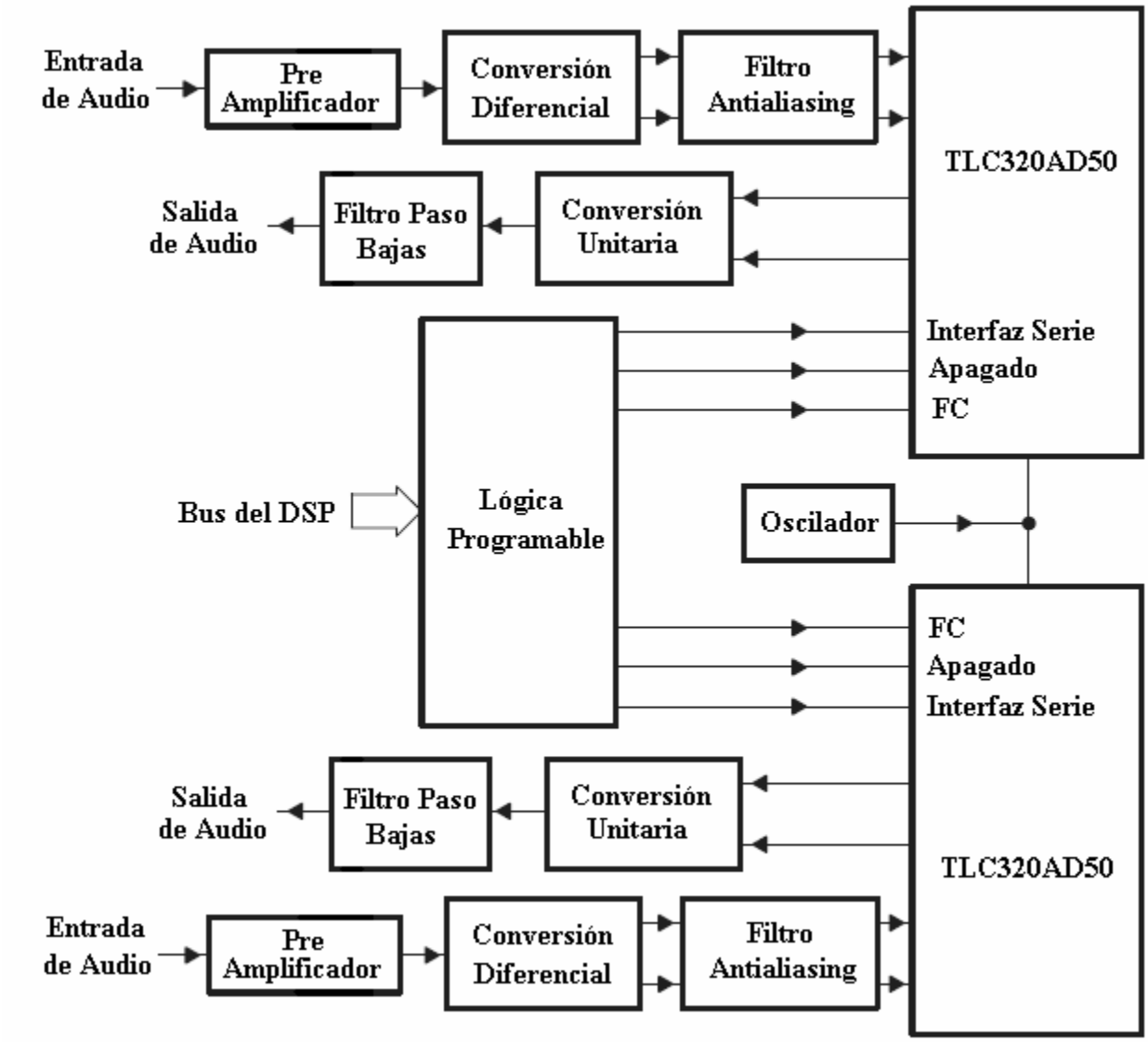

Figura 3.2 Diagrama de Bloques de la Conversión Analógica – Digital, Digital – Analógica (codec) en la Tarjeta DSK

Etapa del Pre-amplificador

Esta etapa del convertidor analógico-digital, ilustrada en la *figura 3.3* es configurable mediante software, utilizándose la entrada marcada como "GANANCIA". Cuando esta entrada es deshabilitada el pre-amplificador tiene una ganancia unitaria; sin embargo, cuando está habilitada, el pre-amplificador tiene una ganancia de 10 dB, a una frecuencia de 20 kHz [2].

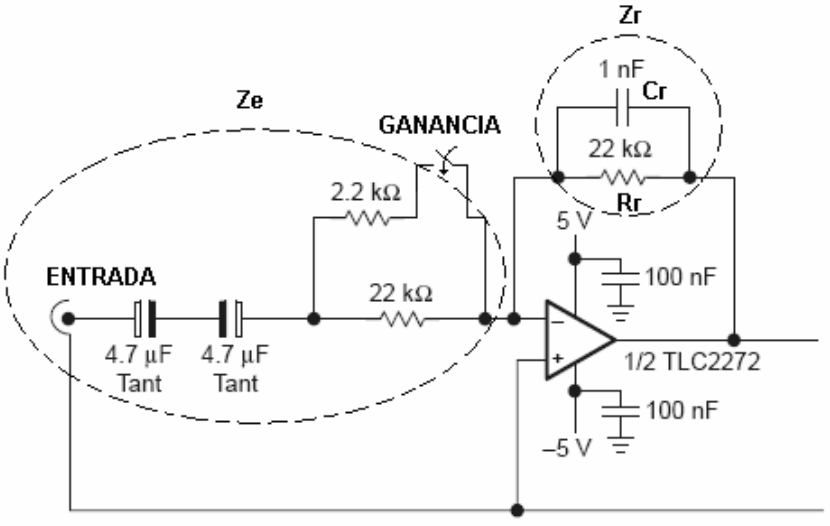

Figura 3.3 Etapa de Pre-Amplificación

El amplificador operacional es el TLC2272 polarizado con +5 V y -5V, el cual tiene una configuración inversora, cuyas ecuaciones son las siguientes:

$$V_{salida} = -V_{entrada}\frac{Z_r}{Z_e} \qquad (3.2)$$

Donde:

Z_r es la impedancia de realimentación.

Z_e es la impedancia de entrada.

La impedencia de realimentación, empleada en la *ecuación (3.2)*, es el resultado del paralelo de la resistencia de realimentación (Rr) y el capacitor de realimentación (Cr), cabe mencionar que la impedancia de un capacitor está definida en función de la frecuencia (*f*) por $Z_C = \frac{1}{2\pi fC}$, entonces:

$$Z_r = \left((R_r)^{-1} + \left(\frac{1}{2\pi f C_r} \right)^{-1} \right)^{-1} = \frac{1}{\frac{1}{R_r} + 2\pi f C_r} = \frac{R_r}{1 + R_r 2\pi f C_r} \qquad (3.3)$$

La impedancia de entrada puede ser de 22 kΩ o de 2 kΩ que resulta del paralelo de las resistencias de 22 kΩ y 2.2 kΩ, esto se debe a la configuración que el usuario determine mediante software [3]. Por lo tanto la ganancia del circuito pre-amplificador está dada por la *ecuación (3.4)*:

$$G = \frac{V_{salida}}{V_{entrada}} = -\frac{\frac{R_r}{1 + R_r 2\pi f C_r}}{Z_e} \qquad (3.4)$$

Calculando con los valores de la *ecuación (3.4)*, a una frecuencia de 20 kHz, tenemos:

$$G[dB] = 20\log\left(\frac{\frac{22x10^3}{1+\left(22x10^3\right)\left(2\pi\right)\left(20x10^3\right)\left(1x10^{-9}\right)}}{2x10^3}\right) = 9.31[dB] \qquad (3.5)$$

Conversión Diferencial.

Debido a que el convertidor analógico-digital TLC320AD50C necesita que la señal de entrada sea de tipo diferencial para su correcto funcionamiento, esta etapa consta de tres amplificadores operacionales y se puede apreciar en la *figura 3.4*. El primer amplificador operacional se polariza con ± 5 V y está configurado de manera inversora, con ganancia unitaria, con el fin de quitar el defasamiento de 180° que tiene la señal a la salida de la etapa pre-amplificadora. Los otros dos amplificadores también tienen ganancia unitaria pero son los encargados de generar la señal diferencial, la cual se logra gracias a la polarización de cada amplificador operacional; cuando el voltaje de polarización es +5 V la señal se satura a ese voltaje positivo, generando la parte positiva de la señal diferencial, mientras que la parte negativa se obtiene al saturarse el amplificador operacional que está a polarizado con el voltaje negativo de -5 V [4].

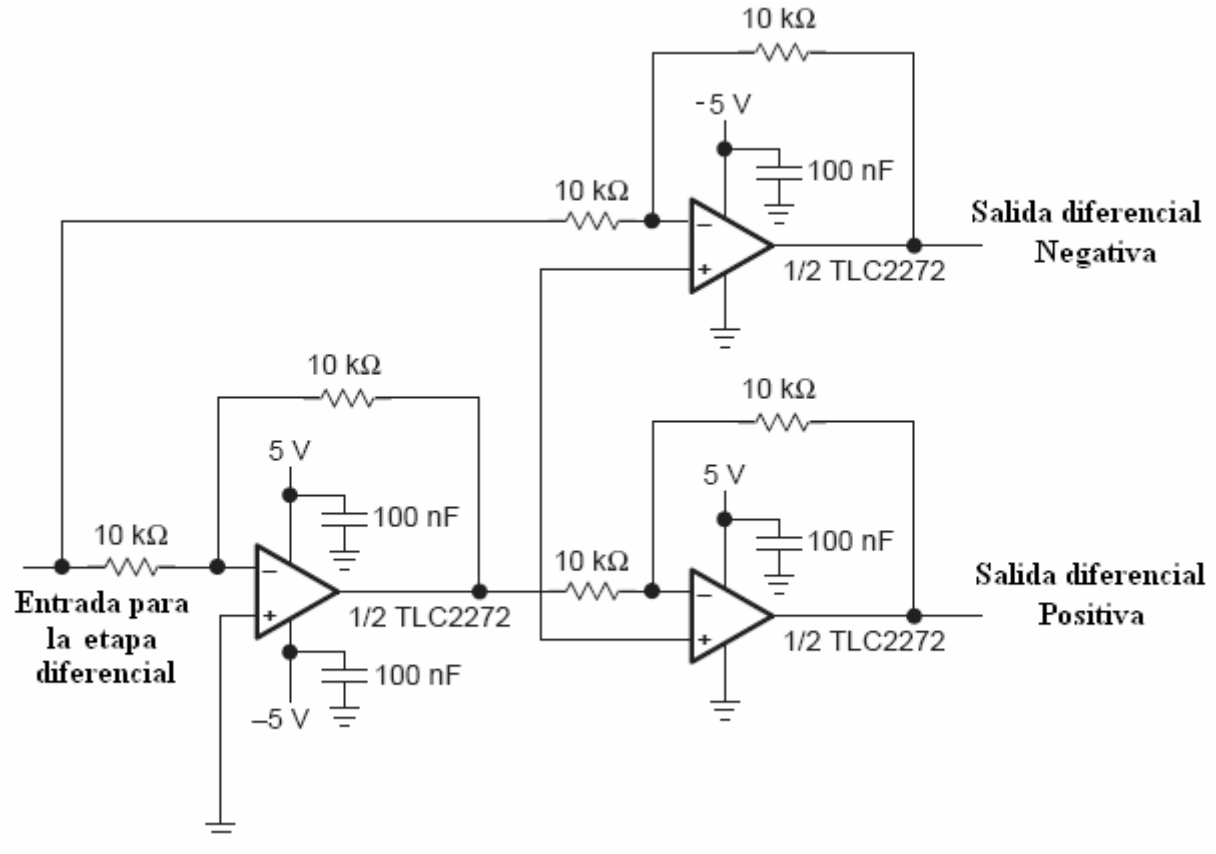

Figura 3.4 Etapa de Conversión Diferencial

Filtro Antialiasing (En la entrada).

La configuración de este filtro se muestra en la *figura 3.5*. Este diagrama representa un filtro paso bajas de primer orden cuya frecuencia de corte esta dada por la *ecuación (3.6)*. Las señales de salida de dicho filtro son las marcadas con INP e INM, que se pueden entender como las entradas "positiva" y "negativa" del convertidor.

$$f_c = \frac{1}{2\pi RC} \tag{3.6}$$

Sustituyendo los valores de la *figura 3.5* se obtiene una frecuencia de corte de 19.92 kHz. Este valor se aproxima al valor teórico máximo de cualquier señal de audio, es decir, a la frecuencia de 20 kHz. El objetivo de este filtro, como su nombre lo indica, es para evitar el aliasing.

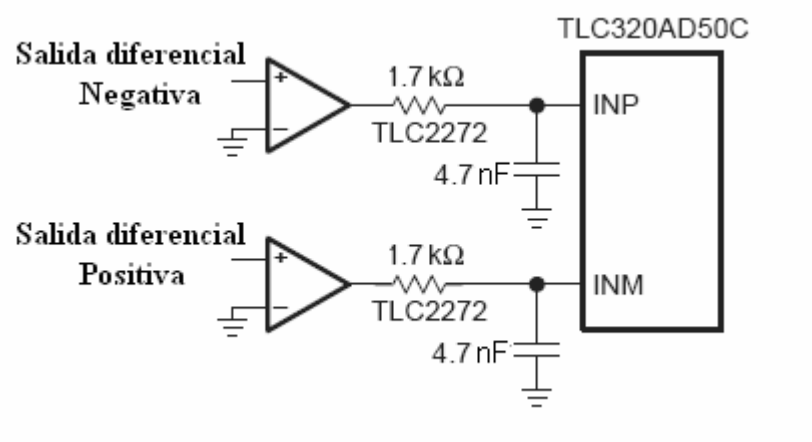

Figura 3.5 Filtro Antialiasing

Acoplamiento Unitario y Filtro Paso Bajas.

Estas etapas se encuentran a la salida del sistema, cuando ya se ha realizado todo el procesamiento. Aquí se tiene una señal diferencial analógica (cuyas salidas se muestran con las etiquetas OUT_P y OUT_M) por tal motivo la señal se hace pasar por una etapa de acoplamiento unitario para referenciar la señal diferencial a tierra [5], como se muestra en la *figura 3.6*.

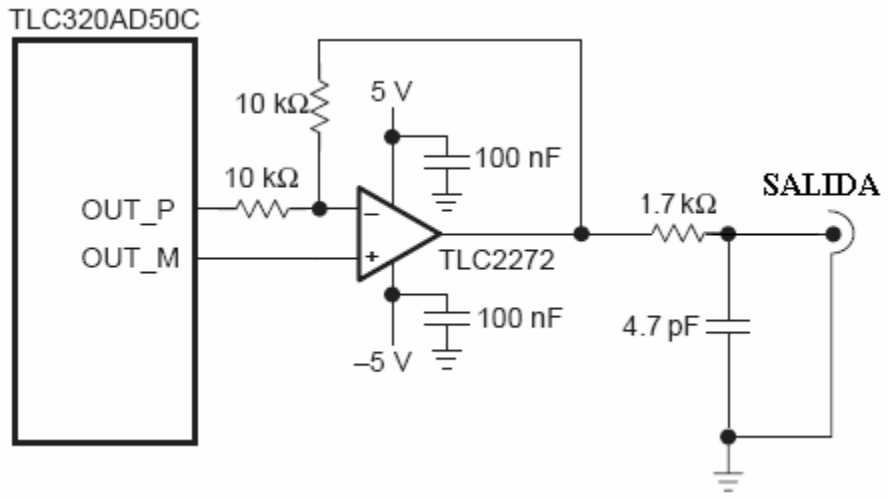

Figura 3.6 Acoplamiento Unitario y Filtro Paso Bajas.

En la *figura 3.6* a la salida se tendrá la señal OUT_P + OUT_M; esta señal entra al filtro paso bajas con frecuencia de corte de 19.92 kHz calculado de acuerdo a la *ecuación (3.6).* Una vez que la señal haya pasado por estas etapas, ya estará acondicionada para ser reproducida por la bocina y así poder percibir el sonido.

La tarjeta DSK tiene integrado el codec TLC320AD50 que proporciona una conversión analógica-digital de alta resolución. Este dispositivo consiste en un par de rutas de conversión síncronas seriales de 16 bits (una para cada dirección). Un convertidor analógico-digital, es un dispositivo electrónico capaz de convertir un voltaje determinado en un valor binario, en otras palabras, este se encarga de transformar señales analógicas a digitales.

Poseen dos señales de entrada llamadas Vref+ y Vref- que determinan el intervalo de voltaje de la de entrada. El dispositivo establece una relación entre su entrada (señal analógica) y su salida (digital) dependiendo de su resolución. Esta resolución se puede saber, siempre y cuando conozcamos el valor máximo que la entrada de información utiliza y la cantidad máxima de la salida en dígitos binarios.

A manera de ejemplo, para un convertidor analógico con la capacidad de convertir una muestra analógica de entre 0 y 5 volts y ancho de palabra de ocho bits, su resolución es:

$$Resolución = \frac{Valor\ Máximo}{2^8 - 1} = \frac{5V}{255} = 0.0196V = 19.6mV \qquad (3.7)$$

Lo anterior quiere decir que el valor de voltaje mínimo que puede registrar el ADC es de 19.6 mV (aproximadamente 20 mV) en el nivel de tensión entre las entradas "Vref+" y "Vref-", y corresponde a un cambio de nivel binario.

Características del codec TLC320AD50.

- Circuito de interfaz analógico de uso general y aplicaciones de audio.
- Interfaz de puerto serial.
- 89 dB de SNR (Relación señal a ruido) para ADC y DAC
- 90 dB de THD (Distorsión Armónica Total) para ADC y DAC
- Frecuencia de muestreo programable.
- Ganancia de entrada y salida programable.
- Formato de los datos en complemento a dos [3].

Los humanos no percibimos información digital directamente; todo proceso de grabación y reproducción de audio tiene en sus extremos la parte analógica. De ahí que se deba realizar una conversión de analógico a digital (DAC por sus siglas en inglés) en tres niveles.

Etapas de un convertidor analógico-digital

Primera. Comprende un filtro conocido como anti-aliasing, el cual define una frecuencia límite. Todos los sonidos por debajo de esa frecuencia, que normalmente es el máximo audible por los humanos, se registrarán; los sonidos que excedan este límite de frecuencia son descartados.

Segunda. Conocida como muestreo, es la toma de valores o voltajes por unidad de tiempo de la señal analógica. El Principio de Nyquist establece que la frecuencia de muestreo debe de ser por lo menos el doble de la frecuencia máxima de la señal de entrada. Para el caso de la voz humana, la frecuencia de muestreo ideal, para no perder información de la onda, debe ser al menos de 8 kHz.

Tercera. Codificación, aquí se asignan valores numéricos a cada registro de la muestra. Como la información digital se apoya sólo en dos dígitos (el cero y el uno), cada punto de la muestra debe expresarse como una potencia de 2. Por ejemplo, sí se usan sólo dos bits para representar numéricamente el valor, sólo habría cuatro valores distintos (00, 01, 10 y 11 que en decimal equivalen a 0, 1, 2 y 3) por lo que la cantidad de sonidos a representar sería muy limitada.

Todo circuito electrónico para su funcionamiento requiere de una fuente de alimentación, que en el caso del convertidor TLC320AD50 es de +5 V de DC para su alimentación. En el caso de algunos amplificadores operacionales que necesitan una entrada de -5 V de DC, este valor negativo de voltaje se obtiene con el circuito inversor de voltaje CMOS ICL7660, cuya configuración se puede observar en la *figura 3.7* [6].

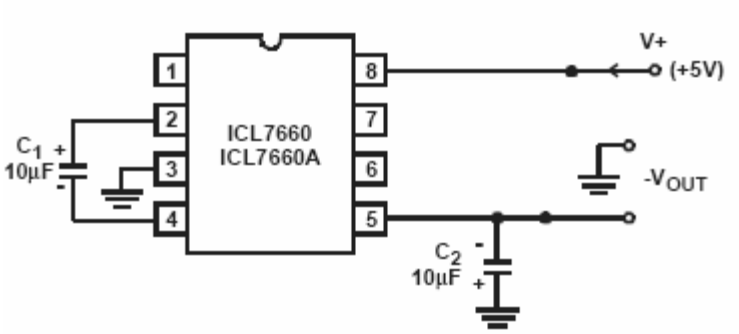

Fig. 3.7 Configuración del Circuito CMOS 7660

El circuito ICL7660 necesita de dos capacitores externos, colocados como se muestra en la *figura 3.9*. El valor recomendado para estos capacitores es del intervalo de 10 μF hasta 100 μF. Este circuito opera de la siguiente manera: el capacitor C_1 se carga con el voltaje V+ para el medio ciclo de trabajo, internamente se intercambian unos interruptores que permiten el flujo de la corriente hacia el capacitor C_2 de tal manera que el voltaje en C_2 es el mismo que en la entrada pero de signo contrario.

Cuando el TLC320AD50C se utiliza en modo independiente, es decir, cuando no está conectado al DSP, dicho convertidor puede trabajar a una frecuencia de 10.24 MHz, ya que a esta frecuencia puede adquirir muestras a una frecuencia de 10, 16 y 20 kHz debido al divisor de frecuencias interno. Sin embargo, cuando es utilizado conjuntamente con el DSP, la frecuencia es limitada por el reloj máximo del DSP5402. En la *tabla 3.2* se pueden apreciar las distintas opciones para configurar el reloj del sistema.

La configuración del reloj del sistema se realiza mediante software, programando los registros de control. El TLC320AD50C cuenta con siete registros de control que son [3]:

- Registro 0. Registro de no operación.
- Registro 1. Registro de Control, los datos en este registro controlan:
    - "Reset" por medio de software.
    - Apagado por medio de Software.
    - Habilitación de las entradas normales o auxiliares.
    - Monitoreo de las entradas normales o auxiliares.
    - Selección del monitor del amplificador de ganancia de salida.
    - Selección del bucle digital.
    - Selección del modo de operación del DAC ya sea en 16 bits o en (15+1) bits.
- Registro 2. Registro de Control, los datos en éste registro son:
    - Contienen el valor de salida de la bandera o *"FLAG"*.
    - Seleccionan el modo de operación telefónico.
    - Contienen la bandera de salida *(output flag),* que indican el desborde del filtro FIR de decimación.
    - Seleccionan el modo de operación del ADC ya sea en 16 bits o en (15+1) bits.
    - Habilitan el bucle analógico.
- Registro 3. Registro de Control, los datos en éste registro:
    - Establecen el número de retrasos SCLK entre la señal de sincronización para envío y recepción de datos (FS) y la señal de sincronización para envío y recepción de datos retrasada (FSD).
    - Informan al dispositivo maestro cuántos elementos esclavos están conectados.
- Registro 4. Registro de Control, los datos en este registro:
    - Establecen el valor de la ganancia tanto de salida como de entrada del amplificador de ganancia.

- Establecen la tasa de muestreo dependiendo del valor seleccionado de N, el cual varía desde N-1 hasta N = 8, en donde por medio de las *ecuaciones (3.8) y (3.9)* se pueden calcular la tasa de muestreo.

$$f_s = \frac{MCLK}{128N} \tag{3.8}$$

$$f_s = \frac{MCLK}{512N} \tag{3.9}$$

Donde MCLK es el reloj maestro del DSP.

- Registro 5. Reservado para pruebas de fabricación, no se debe escribir en este registro.
- Registro 6. Reservado para pruebas de fabricación, no se debe escribir en este registro.

Reg 4, Bit 7	Reg 4, Bits 6–4	8.192 MHz	10.000 MHz	10.240 MHz	11.2896 MHz
bit 7 = 0 PLL on	0 (default)	8 kHz	9.765 kHz	10 kHz	11.025 kHz
	1	(64 kHz)	(78.125 kHz)	(80 kHz)	(88.2 kHz)
	2	(32 kHz)	(39.063kHz)	(40 kHz)	(44.1 kHz)
	3	21.333 kHz	(26.042 kHz)	(26.666 kHz)	(29.4 kHz)
	4	16 kHz	19.531 kHz	20 kHz	22.05 kHz
	5	12.8 kHz	15.625 kHz	16 kHz	17.64 kHz
	6	10.666 kHz	13.021 kHz	13.333 kHz	14.7 kHz
	7	9.1432 kHz	11.161 kHz	11.429 kHz	12.601 kHz
bit 7 = 1 PLL off	0	2 kHz	2.441 kHz	2.500 kHz	2.756 kHz
	1	16 kHz	19.531 kHz	20 kHz	22.05 kHz
	2	8 kHz	9.766 kHz	10 kHz	11.025 kHz
	3	5.333 kHz	6.510 kHz	6.666 kHz	7.35 kHz
	4	4 kHz	4.883 kHz	5 kHz	5.513 kHz
	5	3.2 kHz	3.906 kHz	4 kHz	4.41 kHz
	6	2.666 kHz	3.255 kHz	3.333 kHz	3.675 kHz
	7	0.798 kHz	2.79 kHz	2.857 kHz	3.15 kHz

Tabla 3.2 Configuración del Reloj del Sistema.

El codec está diseñado para interconectar el puerto serial del DSP de la tarjeta DSK, es decir, comparte las señales de reloj y la de los datos. Para evitar la mezcla de información se programa un circuito lógico programable (PLD) en este caso una GAL22V10, que hace la función de un multiplexor y así evitar el cruce de señales.

3.1.3 Procesador Digital de Señales (DSP)

El DSP TMS320VC5402, de Texas Instruments, es un procesador digital de señales de punto fijo, diseñado para la implantación de algoritmos y aplicaciones en tiempo real. Esta construido bajo una arquitectura tipo Harvard modificada con alto grado de paralelismo y bajo consumo de potencia, además de tener modos de direccionamiento versátiles y un conjunto de instrucciones que mejoran su desempeño [7].

Se pueden enumerar algunas características generales del DSP.

- Arquitectura de Multibus avanzado con tres buses separados para memoria dato (16 bits) y un bus para memoria programa.
- Unidad Aritmética y Lógica de 40 bits, incluyendo dos acumuladores independientes de 40 bits y un bloque de corrimiento de 40 bits.
- Multiplicador en paralelo de 17 x 17 bits acoplado a un Sumador dedicado de 40 bits para la operación de Multiplicación – Acumulación (MAC) en un solo ciclo de instrucción.
- Unidad de Comparación, Selección y Almacenamiento (CSSU) para codificación Viterbi.
- Codificador de Exponente para el cálculo del exponente de un valor de 40 bits en el acumulador en un solo ciclo.
- Dos generadores de dirección con ocho registros auxiliares y dos unidades aritméticas de registros auxiliares (ARAU).
- Buses de datos con característica de retención (" Holding").
- Modo extendido de direccionamiento para direccionar hasta 1M x 16 bits.
- 4k x 16 bits en ROM interna.
- 16k x 16 bits en DARAM.
- Instrucciones:
    - Para operaciones de repetición y bloques de repetición de código programa.
    - Para el movimiento de bloques de memoria y para un manejo eficiente de datos y programa.
    - Para operaciones con palabras de 32 bits.
    - Para lectura de dos o tres operandos.
    - Aritméticas con almacenamiento y carga en paralelo.
    - De almacenamiento condicional.
    - De retornos rápidos desde llamadas a interrupciones.
- Periféricos internos:
    - Generador de estados de espera programados por software y un banco de interrupción programable.
    - Generador de reloj con oscilador interno o reloj externo.
    - Circuito de malla de fase enlazada (PLL) programada por software.
    - Dos puertos seriales bufereados multicanal (McBSPs).
    - Interfaz de puerto paralelo de tipo Huésped de 8 bits mejorado (HPI8)
    - Dos Temporizadores de 16 bits con preescalador de 4 bits.
    - Controlador de seis canales de acceso directo de memoria (DMA).

- Control de bajo consumo de potencia con instrucciones IDLE1, IDLE2 e IDLE.
- Emula el estándar 1149.1 (JTAG) de IEEE.
- Velocidad a 100 MHz y período de ciclo de reloj de 10 nseg, y puede realizar hasta 100 millones de instrucciones por segundo (MIPS).
- Maneja seis niveles de pipeline: prebúsqueda, búsqueda, decodificación, acceso, lectura y ejecución.
- Voltaje de operación a 3.3 /1.8 V.

Los sistemas basados en DSP deben trabajar en tiempo real, capturando y procesando información.

Una de las características más importantes de un DSP es su capacidad de realizar operaciones de multiplicación y acumulación (MAC) en sólo un ciclo de reloj. No obstante, es necesario que el dispositivo posea la característica de realizar aplicaciones críticas en tiempo real, lo cual requiere de una arquitectura que soporte un flujo de datos a alta velocidad hacia y desde la unidad de cálculo y memoria. Esta ejecución a menudo requiere el uso de unidades DMA (Direct Memory Acess) y generadores de direcciones duales (DAG) que operan en paralelo con otras partes del chip [7]. Los DAG realizan los cálculos de direcciones, permitiendo al DSP buscar dos datos simultáneamente para operar con ellos en un sólo ciclo de reloj, de tal forma que es posible ejecutar algoritmos complejos en tiempo real.

Es importante para DSP tener un mecanismo efectivo de salto para la ejecución de ciclos ya que el código generalmente programado es altamente repetitivo. La arquitectura permite realizar estos ciclos sin instrucciones adicionales ni demoras, las que al ejecutarse millones de veces empiezan a generar retardos significativos.

Arquitectura del DSP TMS320VC5402.

Las arquitecturas de los computadores actuales están comúnmente clasificadas como RISC (Reduced Instruction Set Computers) y CISC (Complex Instruction Set Computers). Estas últimas tienen un gran número de instrucciones sumamente poderosas, mientras que la arquitectura RISC posee pocas instrucciones y realiza movimientos de datos entre registros en un ciclo de máquina. Hoy en día los computadores RISC comienzan a reemplazar a los CISC, porque se puede alcanzar un rendimiento más alto por medio del uso de un compilador eficiente a través de la ejecución de instrucciones simples en forma ordenada [8].

Un DSP estándar tiene muchos rasgos de una arquitectura tipo RISC, pero son procesadores de propósitos específicos cuya arquitectura está especialmente diseñada para operar en ambientes de alta necesidad de cálculo. Un DSP estándar ejecuta varias operaciones en paralelo mientras que un RISC usa unidades funcionales altamente eficientes que pueden iniciar y completar una instrucción simple en uno o dos ciclos de reloj.

El separar la memoria programa y la memoria dato permite acceso simultáneo a las instrucciones del programa y a los datos teniendo con ello un alto grado de paralelismo. Por ejemplo, se pueden

realizar dos operaciones de lectura y una operación de escritura en un solo ciclo. Por tanto, tal paralelismo permite efectuar en un solo ciclo de instrucción operaciones aritméticas, lógicas y de manipulación de bits. La arquitectura del DSP C5402 se puede observar en la *figura 3.8.*

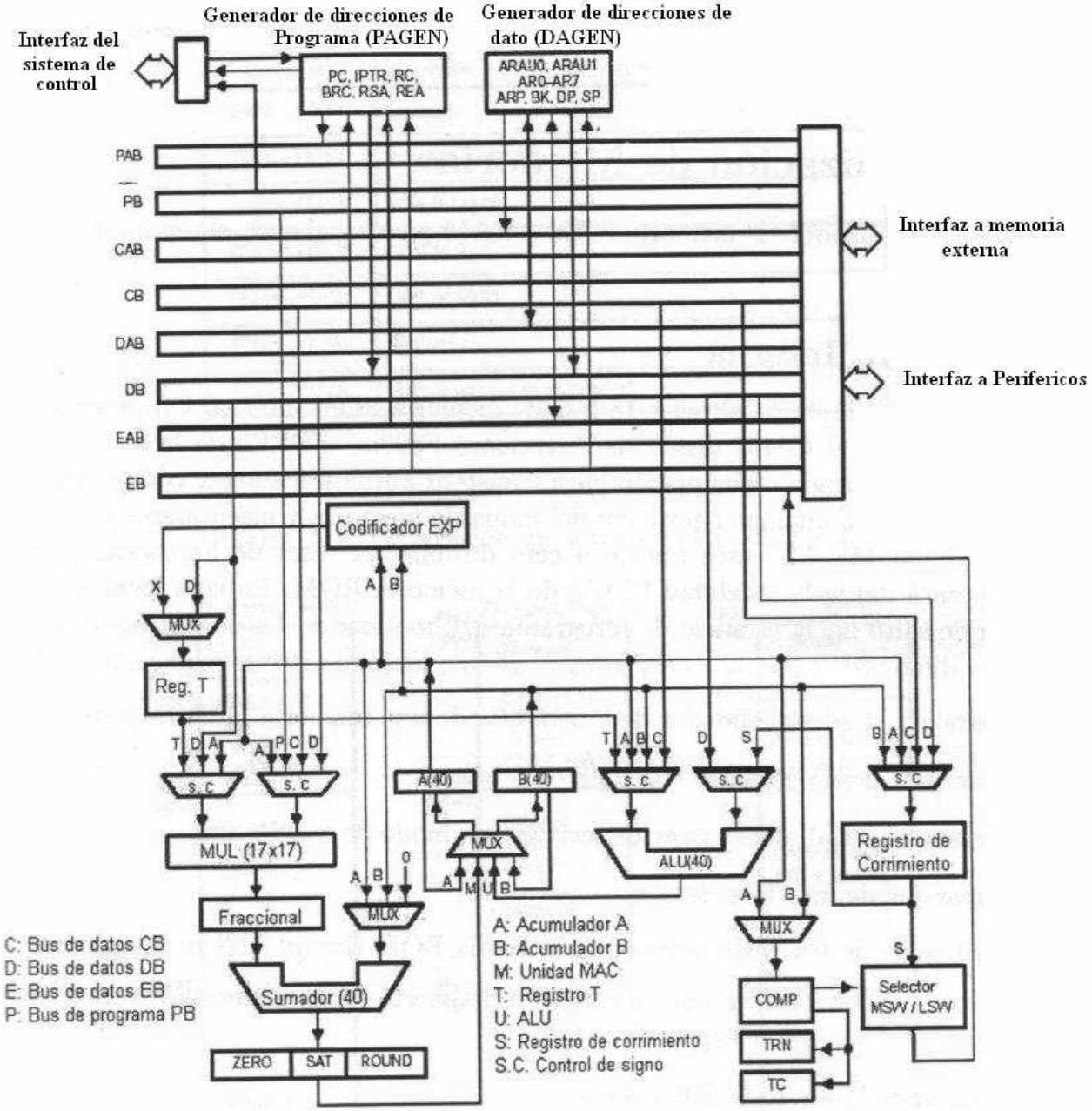

Figura 3.8 Arquitectura del DSP TMS320VC5402

Estructura de Bus.

La arquitectura del C5402 consta de 8 buses de 16 bits; de los cuales 4 buses son para datos y programa y 4 buses para direcciones:

- El bus de programa (PB) transporta el código de instrucción y los operandos inmediatos desde la memoria programa.
- Tres buses de datos (CB, DB, EB) interconectan varios elementos, tales como el CPU, los periféricos internos y la memoria dato.
- Los buses CB y DB transportan los operandos que son leídos desde la memoria dato.
- El bus EB transporta los datos para ser escritos en memoria.
- Cuatro buses de direcciones (PAB, CAB, DAB, EAB) transportan las direcciones necesarios para la ejecución de las instrucciones [7].

Organización de Memoria.

El DSP C5402 dispone de memoria ROM y RAM para ayudar en el rendimiento e integración del sistema.

Memoria ROM Interna

EL C5402 contiene 4k-palabras x 16 bits de memoria ROM interna. Un programa bootloader está disponible para el C5402 desde las direcciones F800h – FBFFh e la ROM. La opción bootloader del C5402 proporciona diferentes maneras para cargar el código alojando varios requerimientos del sistema [7]:

- En paralelo, desde localidades de 8 o 16 bits de una memoria EPROM externa.
- En paralelo, desde espacios I/O de 8 o 16 bits.
- En arranque serial, desde puertos seriales con modo de 8 o 16 bits.
- En arranque de interfaz de puerto huésped.

La distribución de las localidades en la memoria ROM (*tabla 3.3*) es la siguiente:

- Un programa bootloader que se carga desde puerto serie, memoria externa, puertos I/O o interfaces de puertos huésped.
- Una tabla de la ley μ de 256 valores.
- Una tabla de la ley A de 256 valores.
- Una tabla de la función seno de 256 valores.
- Una tabla de vectores de interrupción.

Direcciones	Descripción
F000h – F7FFh	Reservado
F800h – FBFFh	Programa Bootloader
FC00h – FCFFh	Tabla de Ley mu
FD00h – FDFFh	Tabla de ley A
FE00h – FEFFh	Tabla de senos
FF00h – FF7Fh	Reservado
FF80h – FFFFh	Tabla de vectores de interrupción

Tabla 3.3 Disposición de la Memoria ROM Interna.

Memoria RAM Interna

En cuanto a memoria RAM interna, el C5402 contiene 16k palabras x 16 bits localidades del tipo de doble acceso (DARAM). La DARAM está compuesta de 2 bloques de 8k palabras cada uno. Cada bloque puede soportar 2 operaciones de lectura en un ciclo, o una operación de lectura y una de escritura en un ciclo. La DARAM está localizada en el intervalo de direcciones de 0060h – 3FFFh en memoria dato y puede ser mapeada dentro de la memoria programa/dato al poner en uno al bit OVLY (registro PMST).

La distribución del mapa de memoria se muestra en la *figura 3.9*.

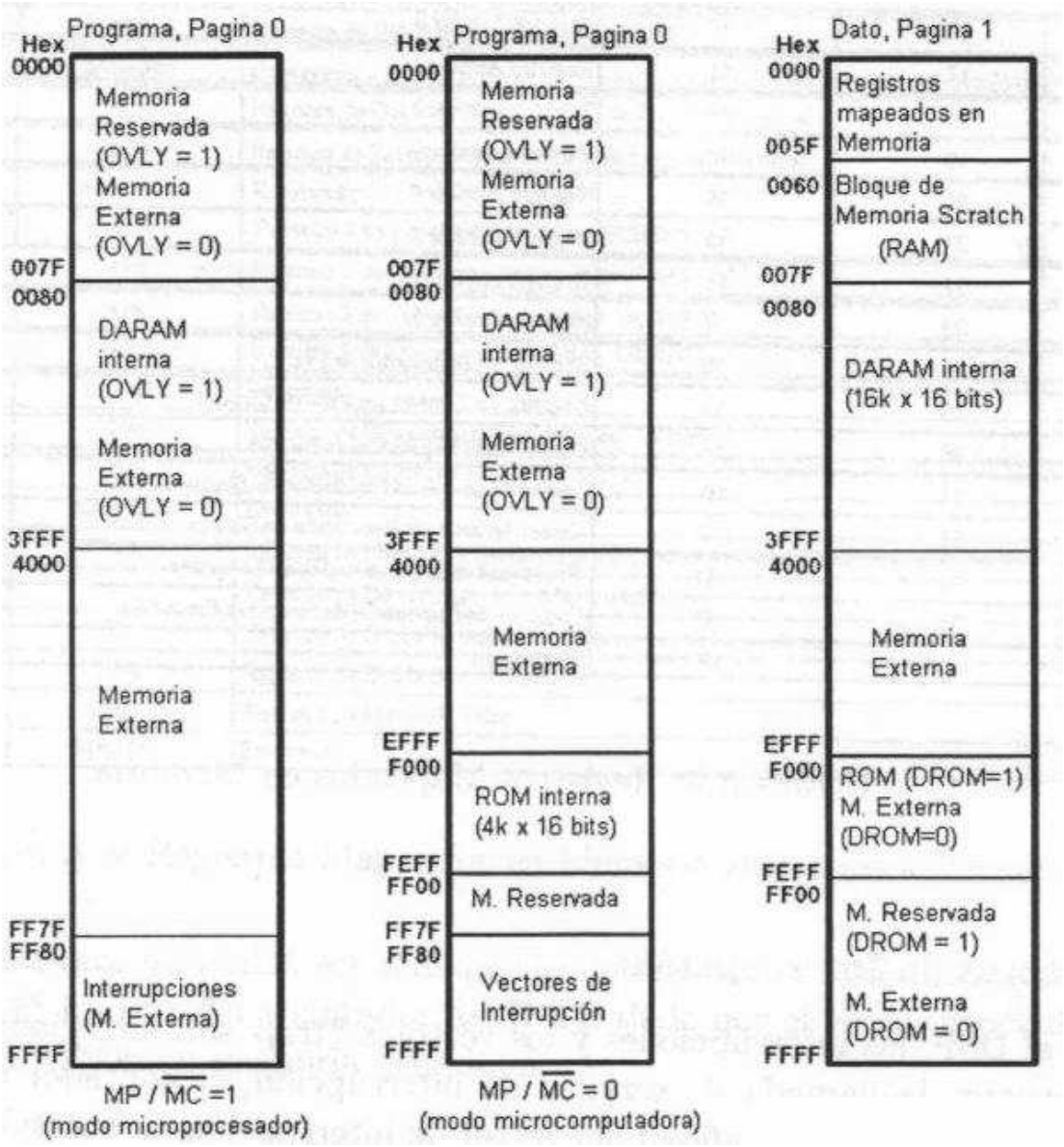

Figura 3.9 Mapa de Memoria.

Tabla de Vectores de Interrupción.

Al resetear el DSP, las interrupciones son direccionadas en memoria programa. Al efectuarse la llamada de servicio de interrupción, el contador de programa (PC) es cargado con la dirección de la localidad del vector de interrupción en cuestión. Por tanto, se reservan cuatro localidades para cada vector de interrupción y de esta manera se efectúa la instrucción de salto con retardo, realizando con ello, la rutina de servicio a la interrupción. Después del reset, las interrupciones son mapeadas a partir de la localidad FF80h en memoria programa. Sin embargo estos vectores pueden ser remapeados para empezar en cualquiera de las páginas definidas en memoria programa, después de haber sido reiniciado el DSP. Lo anterior se realiza cargando en los bits (15-7) del apuntador del vector de interrupción (IPRT) que se encuentra en el registro PMST con la dirección apropiada de la página en cuestión, como se muestra en la *tabla 3.4*.

El reset por hardware no podrá ser remapeado debido a que cuando éste se presente, se cargarán con valor de "1" los bits del apuntador del IPRT dado que el vector asociado al reset siempre está en la localidad FF80h en memoria programa [7].

NOMBRE	DIRECCIÓN		DESCRIPCIÓN
	DEC	HEX	
IMR	0	0	Registro de Interrupción Enmascarable
IFR	1	1	Registro de Bandera de Interrupción
–	2–5	2–5	Reservado para pruebas
ST0	6	6	Registro de Estado 0
ST1	7	7	Registro de Estado 1
AL	8	8	Parte Baja del Acumulador A (15-0)
AH	9	9	Parte Alta del Acumulador A (31-16)
AG	10	A	Bits de Guarda del Acumulador A (39-32)
BL	11	B	Parte Baja del Acumulador B (15-0)
BH	12	C	Parte Alta del Acumulador B (31-16)
BG	13	D	Bits de Guarda del Acumulador B (39-32)
TREG	14	E	Registro Temporal
TRN	15	F	Registro Transitorio
AR0	16	10	Registro Auxiliar 0
AR1	17	11	Registro Auxiliar 1
AR2	18	12	Registro Auxiliar 2
AR3	19	13	Registro Auxiliar 3
AR4	20	14	Registro Auxiliar 4
AR5	21	15	Registro Auxiliar 5
AR6	22	16	Registro Auxiliar 6
AR7	23	17	Registro Auxiliar 7
SP	24	18	Registro de Apuntador de Pila
BK	25	19	Registro de Tamaño para Buffer Circular
BRC	26	1A	Contador de Repetición de Bloque
RSA	27	1B	Dirección Inicial de Repetición de Bloque
REA	28	1C	Dirección final de Repetición de Bloque
PMST	29	1D	Processor mode status (PMST) register
XPC	30	1E	Registro de Paginación de Programa Extendido
–	31	1F	Reservado

Tabla 3.4 Registros Mapeados en Memoria.

Unidad Central de Proceso (CPU)

El CPU está caracterizado por tener:

- Una unidad aritmética lógica (ALU) de 40 bits.
- Dos acumuladores de 40 bits, A y B.
- Un bloque de corrimiento de 40 bits.
- Un Multiplicador / Sumador de 17 x 17 bits.
- Una unidad de comparación, selección y almacenamiento (CSSU).
- Un registro temporal de 16 bits (T).
- Un registro de transición de 16 bits (TRN).
- Un codificador de exponente, para programar aritmética de punto flotante.

Interrupciones.

El C5402 consta de un conjunto de interrupciones tanto internas como externas, además de incluir interrupciones por software y para los periféricos internos. Las localidades de los vectores de interrupción y sus respectivas prioridades para todas las interrupciones internas y externas se muestran en la *tabla 3.5*.

Las interrupciones descritas anteriormente se pueden configurar mediante el registro de interrupciones mascarables (IMR).
De las interrupciones mostradas en la *tabla 3.5* solo se utiliza la de recepción de McBPs1 (BRINT1), como se muestra en el *Anexo 1* (Pág. 104), en donde los registros IMR e IFR se configuran mediante el programa "*sistema.c*" vía instrucciones de ensamblador en "línea".

NOMBRE	LOCALIDAD DECIMAL	LOCALIDAD HEX	PRIORIDAD	FUNCIÓN
RS, SINTR	0	00	1	Reset (Por Hardware y Sotfware)
NMI, SINT16	4	04	2	Interrupción no enmascarable
SINT17	8	08	—	Interrupción por Software #17
SINT18	12	0C	—	Interrupción por Software #18
SINT19	16	10	—	Interrupción por Software #19
SINT20	20	14	—	Interrupción por Software #20
SINT21	24	18	—	Interrupción por Software #21
SINT22	28	1C	—	Interrupción por Software #22
SINT23	32	20	—	Interrupción por Software #23
SINT24	36	24	—	Interrupción por Software #24
SINT25	40	28	—	Interrupción por Software #25
SINT26	44	2C	—	Interrupción por Software #26
SINT27	48	30	—	Interrupción por Software #27
SINT28	52	34	—	Interrupción por Software #28
SINT29	56	38	—	Interrupción por Software #29
SINT30	60	3C	—	Interrupción por Software #30
INT0, SINT0	64	40	3	Interrupción Externa de Usuario #0
INT1, SINT1	68	44	4	Interrupción Externa de Usuario #1
INT2, SINT2	72	48	5	Interrupción Externa de Usuario #2
TINT0, SINT3	76	4C	6	Interrupción Tmer0
BRINT0, SINT4	80	50	7	Interrupción de Recepción McBPs #0
BXINT0, SINT5	84	54	8	Interrupción de Transmisión McBps #0
Reservado (DMAC0), SINT6	88	58	9	Reservado (default) o Interrupción del canal DMA #0. La selección es hecha en el registro DMPREC
TINT1(DMAC1), SINT7	92	5C	10	Interrupción Timer1 (default) o Interrupción del canal DMA #1. La selección es hecha en el registro DMPREC
INT3, SINT8	96	60	11	Interrupción Externa de Usuario #3
HPINT, SINT9	100	64	12	Interrupción HPI
BRINT1(DMAC2), SINT10	104	68	13	Interrupción de Recepción McBPs #1 (default) o Interrupción del canal DMA #2. La selección es hecha en el registro DMPREC
BXINT1(DMAC3), SINT11	108	6C	14	Interrupción de Transmisión McBPs #1 (default) o Interrupción del canal DMA #3. La selección es hecha en el registro DMPREC
DMAC4,SINT12	112	70	15	Interrupción del canal DMA #4
DMAC5,SINT13	116	74	16	Interrupción del canal DMA #5
Reservado	120–127	78–7F	—	Reservado

Tabla 3.5 Interrupciones.

En el *Anexo 1* (Pág. 107) en el programa "*vectors54xx.asm*" se ubican los saltos a subrutinas de atención de interrupción.

3.2 Diseño de los Componentes del Sistema.

Diseño del Efecto Eco.

El eco se produce cuando las reflexiones de un sonido llegan con un retado superior a 50 ms respecto de la fuente original. Un efecto de Eco básico puede ser obtenido simplemente sumando una muestra de sonido del pasado a la muestra de sonido actual.

En el dominio del tiempo discreto se tiene:

$$y(n) = x(n) + Gx(n-d) \tag{3.10}$$

Donde d es el retraso.

El retraso que tendrá la señal para que se aprecie el efecto de Eco es de 50 ms y, además, conociendo la frecuencia de muestreo, se puede determinar el número de muestras que se deberán retrasar, es decir, el valor del retraso en la función eco.

El siguiente diagrama muestra como son afectados los datos de entrada al pasar por la función eco. Cabe mencionar que las primeras 400 muestras pasan sin sufrir alteración, ya que para dichas muestras el valor del retraso es mayor que el valor del número de muestra respectivo. Esto se hace porque no se puede conocer muestras antes del inicio de los datos de entrada ya que es un sistema causal, en otras palabras, no se pueden determinar los valores de DATO(-1), DATO(-2), DATO(-3), …., DATO(-398), DATO(-399), como se muestra en la *figura 3.10*.

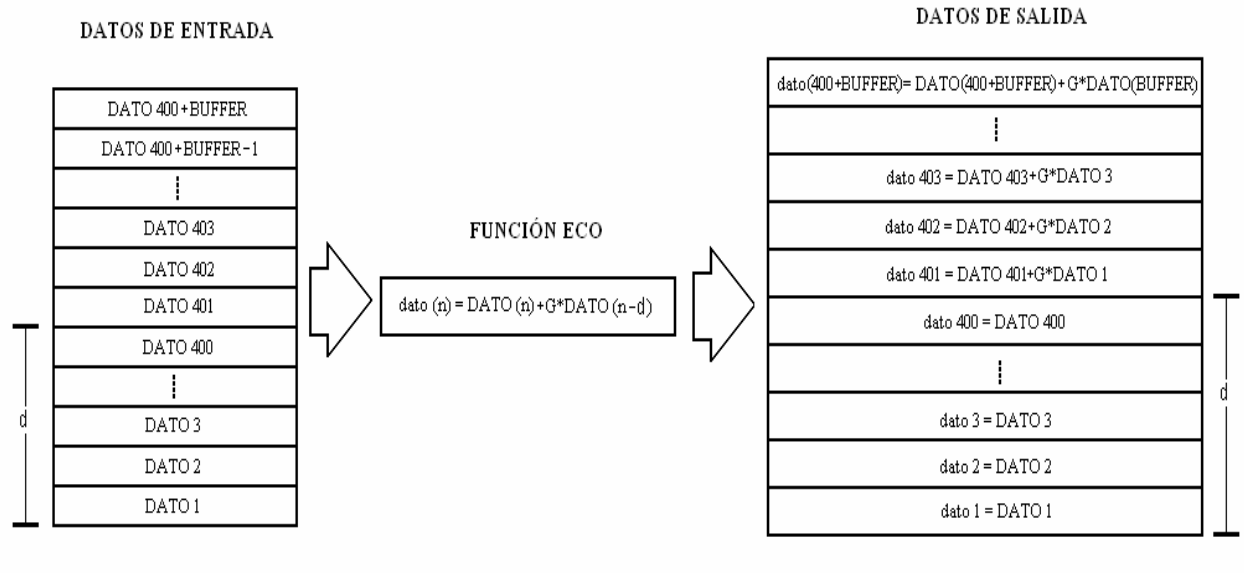

Fig. 3.10 Diagrama del Flujo de Datos de la Función Eco.

Para el diseño de los efectos en Matlab se ocupa un archivo *".wav"*, como señal de prueba, el cual contiene datos0 muestreados a 8000 Hz. Debido a que se necesita 50 ms de retraso y además

conociendo la frecuencia de muestreo, se puede determinar el número de muestras que se deberán retrasar, es decir, el valor del retraso en la función eco de la siguiente manera:

$$d = (50mseg)\left(8000\frac{muestras}{seg}\right) = 400\,muestras \tag{3.11}$$

Entonces, 400 muestras será el número mínimo para poder percibir el efecto Eco y poder asegurar el retraso de 50 ms.

Diseño del Efecto Reverb.

La reverberación es producida en la naturaleza por la reflexión de los sonidos en las superficies, su efecto en cualquier sonido que alcance a la persona que lo escucha depende del ambiente en donde el sonido sea generado. La cantidad y calidad del Reverb depende del volumen y de las dimensiones del recinto y el tipo, forma y número de superficies que el sonido encuentra. Existen diversos caminos por los que el sonido emanado de la fuente puede alcanzar su destino. Debido a que la amplitud del sonido decrementa a un ritmo inversamente proporcional a la distancia viajada, el sonido no solo es retrasado, si no que también tiene una amplitud menor, lo cual hace que el efecto de Reverb tiende a incluir una envolvente de amplitud decreciente [9].

Una vez implementada la función eco, el desarrollo de la función reverb fue sencillo, ya que el efecto reverb se lleva a acabo generando más de un eco en la señal de salida.

Partiendo de la ecuación en diferencias:

$$y(n) = x(n) + G_1(x - d_1) + G_2(n - d_2) \tag{3.12}$$

Los retrasos d_1 y d_2 deben de ser seleccionados adecuadamente para lograr el efecto Reverb, los valores más simples es tomar el valor implementado en la función eco para ser el retraso d_1 y el doble para el retraso d_2, es decir, 50 ms y 100 ms respectivamente.

Diseño del Efecto Flanger.

Este efecto produce un sonido de alta frecuencia que varía rápidamente, sumando a la señal de entrada una copia de si misma que es retrasada una cantidad pequeña pero variable en el tiempo. El retraso debe ser una función del tiempo o de la variable discreta n, para variar de unos pocos milisegundos (10 ms aproximadamente) a 0 para producir el efecto Flanger característico.

Para su implementación digital se utilizan las *ecuaciones (3.13) y (3.14)*:

$$y(n) = x(n) + gx[n - M(n)] \tag{3.13}$$

La variable *g* es conocida también como el parámetro de profundidad (Depth) y determina la proporción de la señal retrasada a la señal de salida, lo que influye directamente en la presencia del efecto Flanger, dicho parámetro varia entre 0 y 1.

El sonido característico del Flanger se manifiesta cuando líneas o ranuras (notches) barren de arriba abajo el eje de frecuencias conforme varía el tiempo, alterando dramáticamente el espectro de la señal procesada [9].

Debido a que el retraso varía con el tiempo el número de muestras retrasadas M debe también variar con el tiempo, esto se logra modulando $M(n)$ con un oscilador de baja frecuencia (LFO), si el LFO es una función senoidal $M(n)$ varia de acuerdo a la siguiente ecuación:

$$M(n) = M_0\left[1 + A sen(2\pi f n T)\right] \tag{3.14}$$

Donde *f* es la frecuencia del Flanger en Hertz, *A* es la amplitud máxima oscilación del retraso y M_0 la longitud promedio del retraso controlando la densidad promedio de líneas o ranuras [9].

Si el efecto flanger es implementado correctamente, M debe de cambiar suavemente con el tiempo y por ello no debe haber brincos en los valores asociados con el redondeo al número entero más cercano. Debido a lo anterior, es necesario hacer un programa o subrutina que redondee los valores fraccionarios que toma $M(n)$.

Para poder crear el efecto Flanger es necesario definir los parámetros de las *ecuaciones (3.13) y (3.14)*. El primero en definirse es el número máximo de muestras retrasadas que se utilizaran.

$$N_{\max} = (10ms)\left(8000\frac{muestras}{s}\right) = 80\,muestras \tag{3.15}$$

Debido a que los valores redondeados de $M(n)$ varían muy poco entre si, se optó por hacer variar más rápido el retraso y a la vez hacerlo dependiente del número máximo de muestras retrasadas, esto se logra multiplicando el retraso redondeado por la décima parte de N, para obtener el número de muestras retrasadas a cada instante. El parámetro de oscilación del retraso resultó en $A = 4$, $M_0 = 1$ y el parámetro $g = 1$ para obtener la máxima presencia del efecto flanger en la señal de salida. Como la frecuencia del Flanger debe de ser pequeña se estableció en $f = 0.5\,Hz$.

Diseño del Efecto Silenciar.

Este efecto como se mencionó en el *Capítulo 2*, se obtiene eliminado muestras de la señal de entrada de audio, dejando solo audibles las muestras que permanecen dentro de la señal del efecto silenciar. No existe una fórmula o un desarrollo matemático para el desarrollo de este efecto, por lo que se tuvo que idear la manera para crear este efecto de manera digital

Observando el diagrama de bloque de la *figura 3.11* se puede determinar que el efecto es sencillo en su implementación, sin embargo, la ganancia G(n) del sistema es variable en el tiempo. Es

necesario determinar la función G(n) que rige la variación de la ganancia del sistema tomando en cuenta que dicha función debe asegurar el principio básico del efecto que es eliminar muestras de la señal de entrada. La manera más simple de lograrlo es teniendo una señal de ganancia que tenga solo dos valores posibles, en otras palabras, cuando G(n) tenga el valor de 0 eliminara las muestras de la señal de entrada; y cuando G(n) tenga el valor de 1 las muestras no serán afectadas en su amplitud y permanecerán audibles.

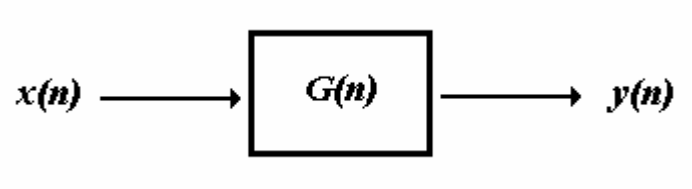

Fig. 3.11 Diagrama de Bloque de la Función Silenciar.

La señal G(n) debe ser periódica para garantizar la eliminación repetitiva de muestras de señal a procesar, por lo que dicha señal de ganancia debe generarse a partir de una señal periódica. Teniendo la amplitud de la señal de ganancia definida, solo resta definir su frecuencia, la cual debe ser de un valor tal que el efecto sea apreciable y además debe ser fácil de modificar. Ahora bien, la frecuencia de la señal de ganancia determina cuántas muestras de la señal son eliminadas y cuántas permanecen audibles, sin embargo, es mejor definir la frecuencia de la señal de ganancia a partir del número de muestras eliminadas que al ser periódica la señal es igual al número de muestras audibles. Entonces se define el número de muestras a eliminar o silenciar y la variable N_S representa la suma de las muestras eliminadas y las audibles, entonces la frecuencia de la señal de ganancia queda definida por:

$$f_C = \frac{f_S}{N_S} \tag{3.16}$$

Donde *fc* es la frecuencia de la señal de ganancia y *fs* es la frecuencia de muestreo.

3.3 Resumen

En este capítulo se describieron las partes y etapas más importantes del sistema. Se determinó que la tarjeta DSK cuenta con muchos dispositivos, pero los utilizados en este trabajo son el convertidor analógico-digital y el DSP; además del micrófono y la bocina. Cada una de estas etapas está diseñada para cualquier señal de audio, ya que cuenta con un ancho de banda de 20 kHz.

Todas las etapas del convertidor analógico-digital TLC320AD50 están implementadas con amplificadores operacionales en configuración inversora y para el caso de los filtros con un circuito RC. Finalmente el DSP cuenta con 144 pines, de los cuales resaltan para esta aplicación los relacionados con las interrupciones, el flujo de datos y el puerto McBSP1 que se comunica con el codec.

Referencias

[1] Escobar S. L., Psenicka B. y Molero A. *Arquitectura de DSPs, familias TMS320C54x y TMS320C54XX y aplicaciones.* Facultad de Ingeniería, UNAM, Enero 2005.

[2] *Evaluation Board for the TLC320AD50C DSP Analog Interface Circuit SLAU039*, Texas Instruments, 2000.

[3] *TLC320AD50C Data Manua*l *SLAS131*, Texas Instruments, 2000.

[4] *Data Acquisition Data Book SLAD001*, Texas Instruments, 2000.

[5] *Operational Amplifiers Data Book Volume A SLYD011*, Texas Instruments, 2000.

[6] *Operational Amplifiers Data Book Volume B SLYD012*, Texas Instruments, 2000.

[7] Molero A. Miguel y Barajas P. Gerardo, *Síntesis de voz en tiempo real empleando una arquitectura DSP* Tesis de Licenciatura, FI, UNAM, México 2004.

[8] Lynn Fuerst, *Introductory Digital Signal Processing with Computer Applications*, USA, 2006.

[9] Smyth T. CMPT 889: *Lectura 7 Delay Effects: Flanging, Phasing, Chorus, Artificial Reverb. School of Computing Sciencie*, Simon Fraser University, 2005.

CAPÍTULO 4.
IMPLEMENTACIÓN DEL SISTEMA.

La finalidad de este capítulo es mostrar el proceso mediante el cual se implementó el Sistema de Efectos Especiales de Audio en Tiempo Real, incluyendo los pasos básicos para la creación de proyectos en el DSP utilizado, como son diseñar los programas fuente, ligar dichos archivos y la configuración de los diversos parámetros de funcionamiento del DSP. Además se muestran las rutinas implementadas para lograr el funcionamiento de cada uno de los efectos de audio en tiempo real, así como el funcionamiento de las rutinas a nivel hardware.

Una vez que se tiene el sistema completo depurado, se muestra el procedimiento de grabar los programas del sistema en memoria flash, para tener un sistema completamente independiente.

4.1 Realización de los Programas.

Teniendo claro nuestro problema a resolver, que es la generación de **efectos especiales de audio** y conociendo lo suficiente el hardware sobre el que se va a implementar la aplicación, en este caso el DSP TMS320C5402 y la tarjeta de desarrollo DSK, se procedió al diseño y realización de los programas.

Como metodología de prueba y simulación se utilizó Matlab, para tener un conocimiento del funcionamiento de los efectos de audio. Una vez teniendo los programas elaborados en Matlab, se trasladaron a un lenguaje de programación reconocido por el DSP. En este caso se realizó la programación en lenguaje *C*, se establecieron las modificaciones necesarias para que los programas funcionaran en tiempo real.

Debido a que la programación en instrucciones simples de Matlab es muy parecida a *C*, las modificaciones más severas se realizaron para obtener el desempeño en tiempo real del sistema. Dichos cambios se realizaron individualmente para cada efecto especial, teniendo así un proyecto en el software de desarrollo para el DSP, para cada efecto, con la intención de unir dichos proyectos en uno solo posteriormente.

4.2 Creación de los Proyectos en el Code Composer Studio.

Para crear un proyecto en el ambiente de desarrollo Code Composer Studio (CCS), se necesitan varios tipos de archivos, los archivos fuente ("*.asm*" y "*.c*"), los archivos de comandos ("*.cmd*") y librerías ("*.lib*" y "*.h*"), que al construir el proyecto originan un archivo se salida ("*.out*"). Los proyectos creados individualmente comparten muchos de los archivos mencionados anteriormente, a excepción del archivo "*.c*", que contiene todas las variables y funciones de cada uno de los efectos. En la *figura 4.1* se muestra un diagrama de la generación de un proyecto en el CCS.

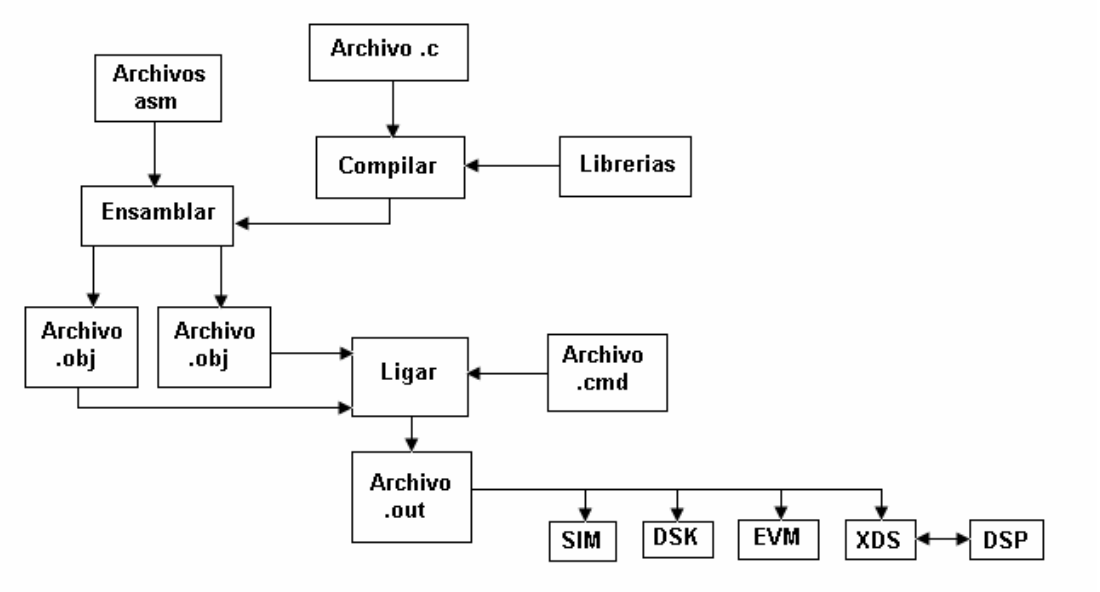

Figura 4.1 Generación de un Proyecto.

Para realizar un programa en el CCS lo habitual es la creación de un proyecto en el cual se tienen que realizar los siguientes pasos para su ejecución:

- Creación del proyecto.
- Creación de los archivos de código fuente, en lenguaje ensamblador y lenguaje *C*.
- Especificar la localización de los ficheros "*include*" y las librerías a utilizar.
- Compilar y ensamblar, configurar opciones de compilado tales como indicar conflictos de pipeline, optimizar espacio memoria y visualizar problemas de traslape en memoria de variables, reservar registros auxiliares para su uso específico en archivos en *C*, etc.
- Ligar los diversos archivos con códigos fuente con un archivo de comandos con extensión "*.cmd*" donde se definen los espacios de memoria permisibles del procesador digital de señales (DSP).
- Cargar el proyecto compilado en el DSP [1].
- Correr la aplicación.

Para la creación del proyecto se escoge la opción "*Project*" de la barra de herramientas, después seleccionar el tópico "*New*", de tal manera que se abrirá un cuadro de dialogo donde nombrará al proyecto con extensión "*.mak*" [1]. Este procedimiento se muestra en la *figura 4.2*.

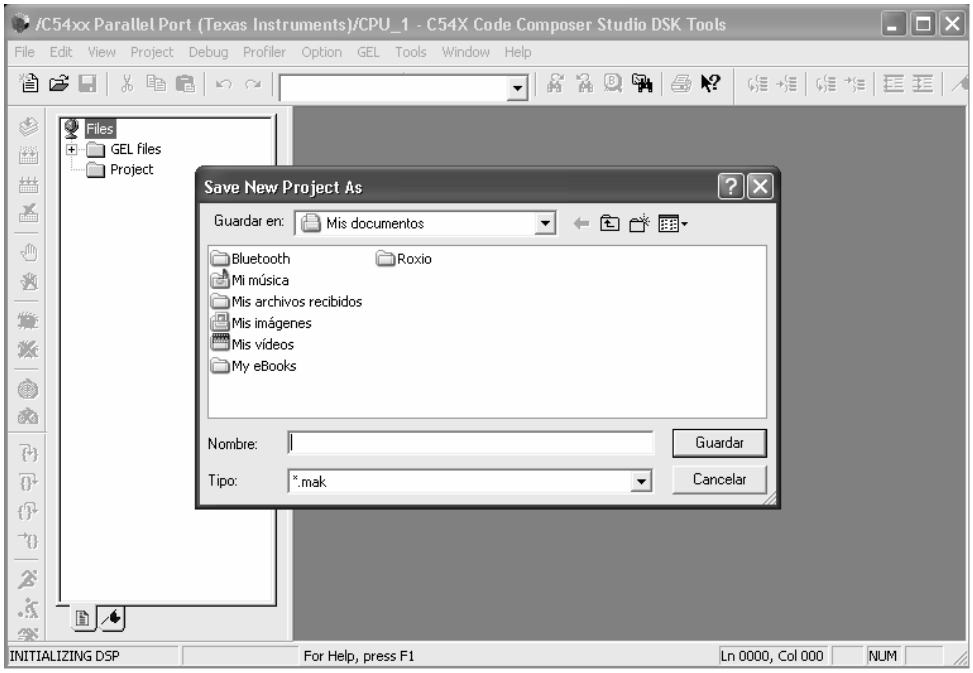

Figura 4.2 Creación de un Proyecto en el CCS.

De igual manera para la creación y edición de de los códigos fuente se escoge la opción "*File*" de la barra de herramientas con la selección de "*New*" donde se nombrará al código con la extensión debida ("*.asm*", "*.c*", "*.cmd*"), lo cual se muestra en la *figura 4.3* [1].

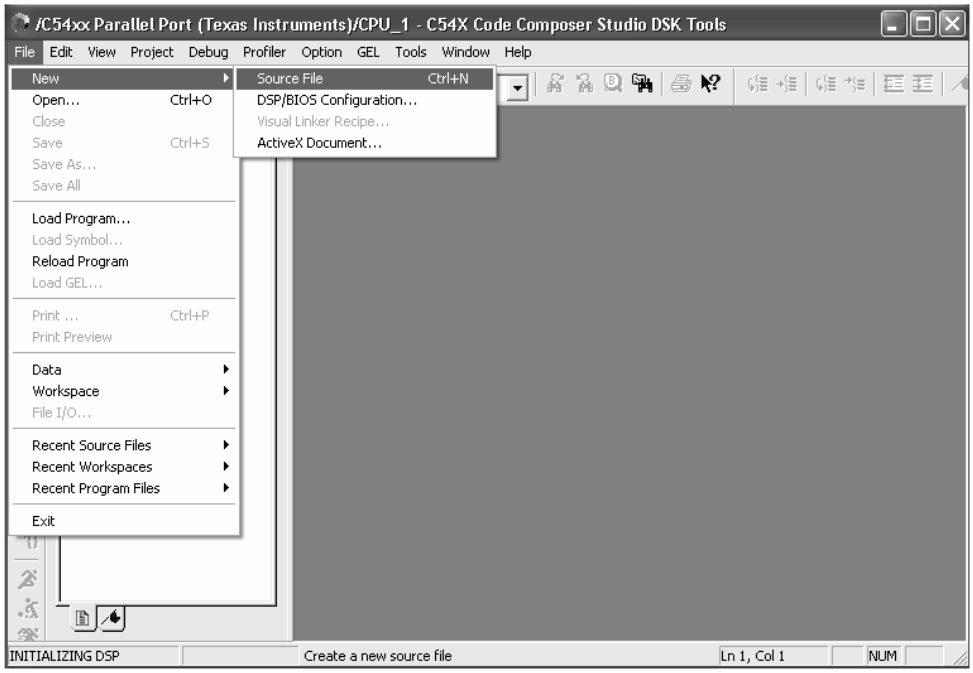

Figura 4.3 Creación de un Archivo en el CCS.

El primer paso para poder implementar los proyectos por cada efecto especial en el DSP fue **crear un archivo .*C*** en el cual se especificaron las instrucciones adecuadas para que el DSP pudiera recibir una señal de audio de entrada digitalizada a fin de procesarla, además dicho programa proporcionará una señal de salida de audio, que en un principio fue la misma señal de entrada, a fin de poderla conectar a unas bocinas para la salida de audio de la tarjeta DSK como se muestra en la *figura 4.4*. En la *figura 4.5* se muestra un esquema de los diferentes archivos y herramientas de generación de código utilizadas.

Dicho programa tiene como base una **rutina de interrupción** que se presenta con la **recepción** de una muestra de la **señal de entrada**, y transmite esa muestra como parte de la **señal de salida**, para lo cual es necesario definir los parámetros de la conversión analógica digital, y de la conversión digital analógica para la salida del sistema. Dichos parámetros fueron los siguientes:

- Codificación para la conversión analógica digital a 15 Bits.
- Codificación para la conversión digital analógica a 15 Bits.
- Ganancia de entrada 0 dB.
- Ganancia de salida -6 dB.
- Frecuencia de muestreo $F_S = 8000$ Hz.

Además se incluyeron las instrucciones para habilitar interrupciones así como definir sus vectores, y los registros de recepción y transmisión. Se incluyeron las librerías necesarias como son: "*type.h*", "*board.h*" y "*codec.h*", donde se encuentran las definiciones de las funciones del codec. Los vectores de interrupción se definieron en el archivo "*vectors54xx.asm*".

Los parámetros fueron definidos mediante el siguiente código:

```
#define GLOBAL_INT_ENABLE     asm( " rsbx intm ")// Definición de  Habilitación de
                                                 // Interrupción.

#define GLOBAL_INT_DISABLE    asm( " ssbx intm ")// Definición de deshabilitación de
                                                 // Interrupción.

GLOBAL_INT_DISABLE;                              // Habilitación de Interrupción.

    brd_init(100);                            // Habilitar Tarjeta para trabajar a 100Mhz.

    hHandset = codec_open(HANDSET_CODEC);        // Abrir Codec.

    codec_dac_mode(hHandset, CODEC_DAC_15BIT);   // DAC 15 Bits.

    codec_adc_mode(hHandset, CODEC_ADC_15BIT);   // ADC 15 Bits.

    codec_ain_gain(hHandset, CODEC_AIN_0dB);      // Ganancia de entrada 0dB.

    codec_aout_gain(hHandset, CODEC_AOUT_MINUS_6dB); // Ganancia de Salida -6dB.

    codec_sample_rate(hHandset,SR_8000);       // Frecuencia de muestreo fs=8000Hz.
```

En la *figura 4.4* se muestra un diagrama general del hardware involucrado en el sistema.

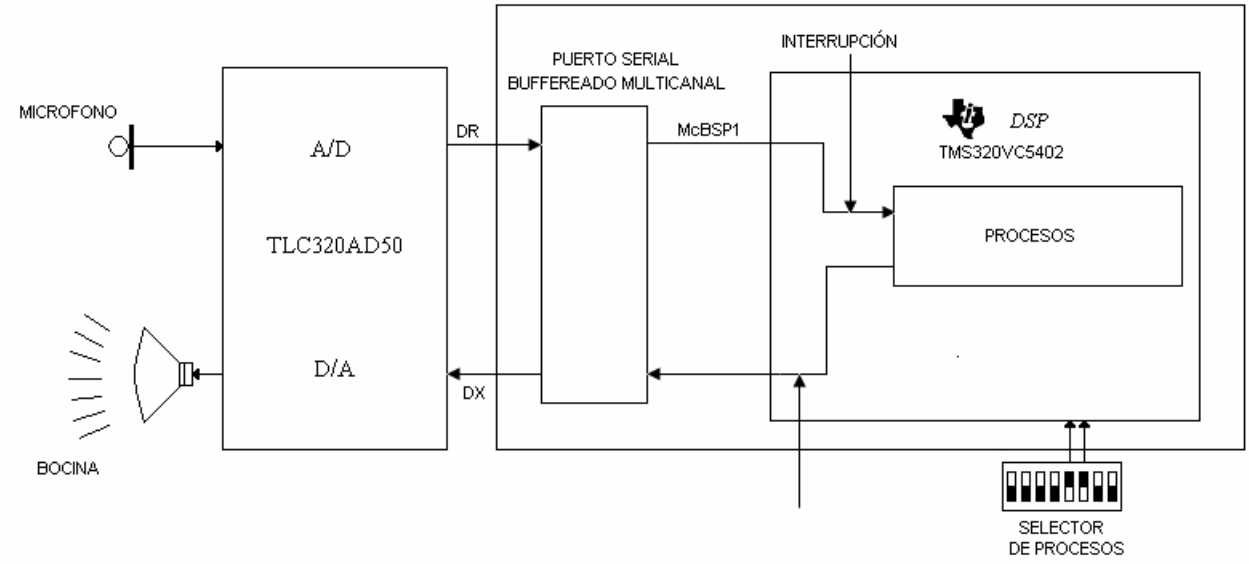

Figura 4.4 Hardware del Sistema.

El siguiente **archivo** que se elaboró fue el de **ligado de comandos**, el cual define el espacio de la memoria que se asigna a datos, programa e interrupciones. El archivo fue llamado "*ligador.cmd*".

Una vez teniendo todos los archivos anteriores se **creó un proyecto** "*lad_da.mak*", que conjunta los archivos mencionados además de los archivos "*rts.lib*", "*drv5402.lib*" y "*dsk5402.lib*" necesarios para el control del funcionamiento del DSP y la tarjeta DSK.

El siguiente paso fue compilar y ensamblar los archivos fuente que en este caso son los de extensión "*.asm*" y "*.c*". Una vez que el compilador mostró que los archivos fuente no tienen errores se procedió a construir el proyecto, esto es, ligar todos los archivos contenidos en el proyecto para tener un solo archivo de salida el cual es el que se carga al DSP y tiene la extensión "*.out*".
Los procesos de compilado y construcción se realizan mediante las herramientas de generación de código, las cuales se muestran en la *figura 4.5*.

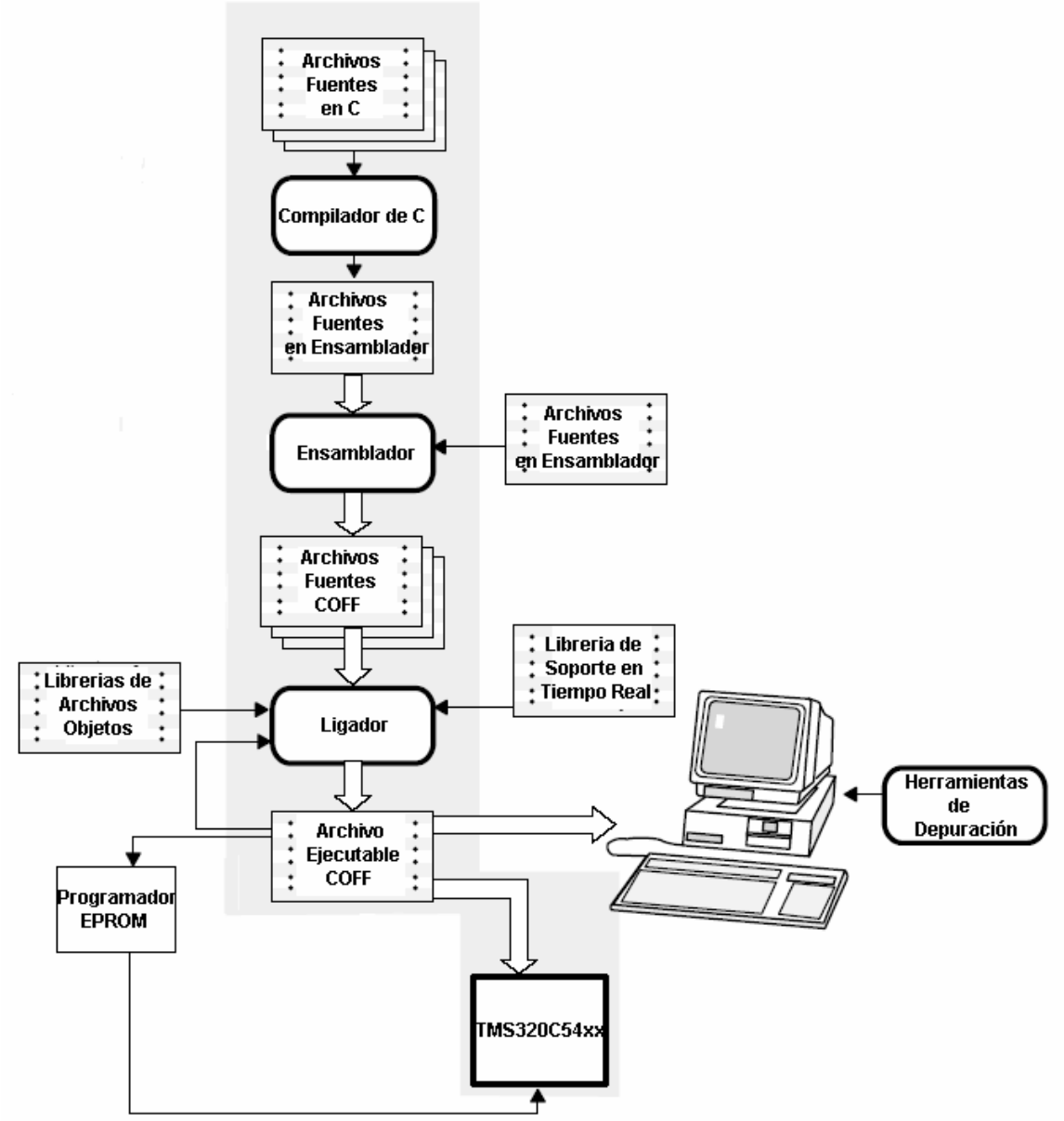

Figura 4.5 Diagrama a Bloques de las Herramientas para la Generación de Código.

Las herramientas para la generación de código permiten configurar los programas mediante un proceso de compilación, ensamblado, ligado y referencias a librerías a usar. Dicho proceso puede realizarse de manera transparente al usuario gracias a las opciones de compilación del proyecto del ambiente CCS, o bien, efectuarlo de modo manual. A continuación se da una breve descripción de las herramientas para la generación de código [1]:

- El compilador de lenguaje C acepta código fuente en C produciendo código fuente en lenguaje ensamblador nativo del DSP.
- El programa ensamblador traslada archivos o códigos fuente escritos en lenguaje ensamblador a archivos objeto de lenguaje máquina, donde el lenguaje máquina está basado en el estándar de formato de archivos de objetos comunes (COFF).

- El ligador combina archivos objetos en un módulo único de objetos ejecutables. Al crear este módulo ejecutable se realiza una relocalización e inserción de las referencias externas al proyecto. El ligador de archivos permite conjuntar los diferentes archivos del proyecto en un archivo único en modo de librería.
- La utilidad de traducción de mnemónicos de lenguaje ensamblador a modo algebraico convierte el código escrito en lenguaje ensamblador a un código descrito de forma algebraica (instrucciones algebraicas) y viceversa.

Utilizando las herramientas del CCS y realizando los procedimientos explicados e indicados en la *figura 4.5*, se realizó un proyecto piloto de entrada y salida con el fin de probar su correcto funcionamiento en tiempo real, que en este caso sería el oír a la salida del DSP la misma señal de entrada. Se activo la opción "*Run*" y se alimentó una señal a la entrada, a la salida se conectaron unas bocinas y se comprobó que el sistema funcionaba correctamente.

Una vez funcionando el proyecto de prueba anterior, se procedió a modificarlo para implementar uno a uno los efectos especiales de audio, teniendo como primera meta que los efectos funcionaran en tiempo real además de que distinguiera diferencia entre la señal original y la señal con efecto.

Una vez teniendo todos los efectos de audio funcionando en tiempo real es necesario depurarlos para su correcto funcionamiento, esto es, ajustar sus diferentes parámetros con el fin de obtener un desempeño óptimo de los mismos. Dicho desempeño debe incluir; obviamente funcionamiento en tiempo real, distinción entre la señal original y la señal con efecto, distinción del mensaje de la señal original en la señal con efecto, señal con efecto libre de ruido y distorsión.
Dichos parámetros fueron en principio establecidos conforme a los valores que se definen en la teoría de cada uno de los efectos, sin embargo estos valores son solo puntos de partida que nos indican en que intervalo de valores deben encontrarse dichos parámetros. Los valores definitivos se establecieron a partir de la experimentación que se realizó con el funcionamiento de cada uno de los efectos.

Eco.

El primer efecto en implementarse fue el Eco debido a que es uno de los más sencillos en su realización. Partiendo de la ecuación en diferencias que lo define:

$$y(n) = x(n) + Gx(n-d) \tag{4.1}$$

Donde d se refiere al retraso de la señal, y G es la ganancia o amplitud de la señal retrasada.

El primer paso es definir el retraso en milisegundos que tendrá la señal para que se aprecie el efecto de Eco y después convertirlo a un número entero de muestras discretas. El retraso utilizado inicialmente es de $50\,mseg$, con base a la frecuencia de muestreo, el retraso d en muestras se calcula:

$$d = (50mseg)\left(8000\frac{muestras}{seg}\right) = 400\,muestras \qquad (4.2)$$

Para resolver el problema del funcionamiento en tiempo real se definió un buffer de memoria para una ventana de la señal de entrada, el cual debe ser al menos de 400 localidades de memoria como se muestra en la *figura 4.6*.

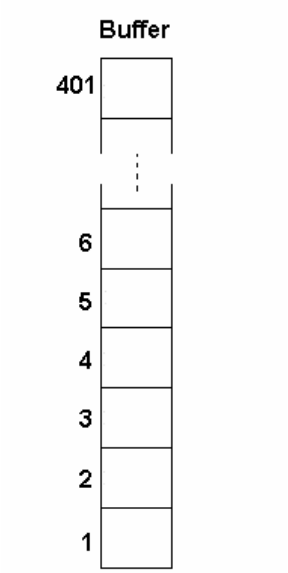

Figura 4.6 Buffer para el Efecto Eco.

Se realizó una función llamada "*eco*", la cual es llamada cada vez que ocurre una interrupción de entrada de una muestra, dicha función en principio debe guardar la muestra actual de la señal de entrada al final del buffer para ser utilizada posteriormente como la señal retrasada, una vez hecho esto la función suma a la muestra actual la muestra localizada a una distancia d en el buffer, en este caso en la posición 1, lo cual es "$x(n)+Gx(n-d)$" establecido en la ecuación en diferencias para el Eco. Por último, antes de regresar el valor mencionado anteriormente, la función realiza un corrimiento en el buffer de muestras de la señal eliminando la última muestra y recorriendo las demás un lugar en el buffer para permitir almacenar la siguiente muestra que llegue en la próxima interrupción, este proceso es mostrado en la *figura 4.7* y *4.8.*

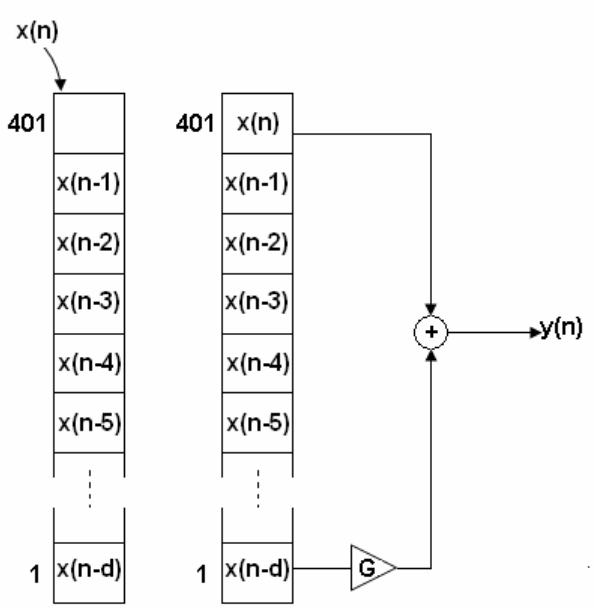

Figura 4.7 Generación del Efecto Eco a partir del Buffer de Memoria.

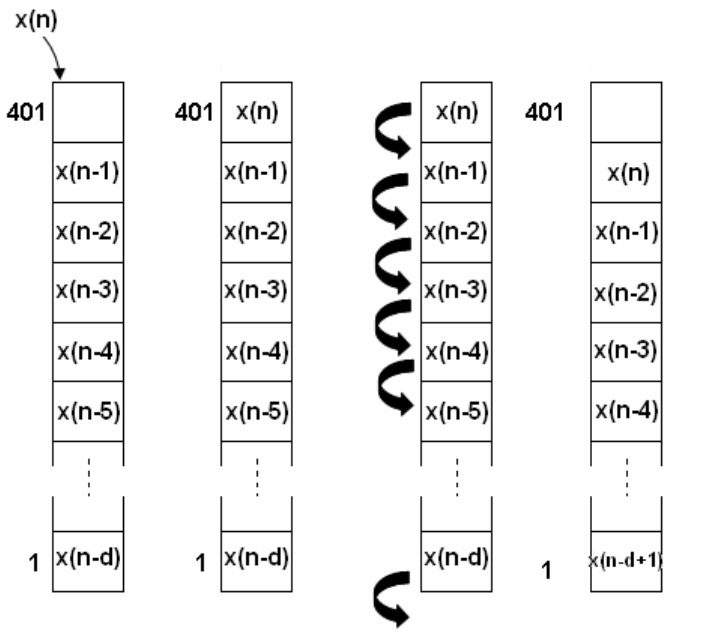

Figura 4.8 Corrimiento de Muestras en Memoria de la Función "*eco*".

El procedimiento de corrimiento se realiza con el código:

```
for (i=N; i>-1; i--)                    // Barrido del Buffer de memoria.

          {

          buff[N-i]=buff[N-i+1];        // Corrimiento de muestra un lugar en el buffer.
          }
```

Donde *N* es la longitud del buffer.

Ajustes al Efecto Eco.

Al principio con el valor de retraso el efecto Eco era poco apreciable, ya que se oye la señal retrasada casi al mismo tiempo que la señal original, de acuerdo a las pruebas realizadas el retraso debe ser mayor a $50\,mseg$, por otro lado al ir aumentando el retraso del Eco se llegó a observar que entre mas grande es el retraso, menos se distingue el mensaje de la señal de audio, resultando entonces que el retraso no debe ser mayor a $200\,mseg$. Ahora bien el valor definitivo del retraso debe cumplir ambas condiciones, haciendo pruebas de valores de retraso entre los límites, se concluyó que el retraso adecuado para el efecto Eco es de $100\,mseg$ esto es:

$$d = (100mseg)\left(8000\frac{muestras}{seg}\right) = 800\,muestras \qquad (4.3)$$

Por lo que el ajuste de este efecto se reduce a definir un buffer para la señal de entrada de 800 localidades de longitud, teniendo siempre en la localidad 1 la muestra retrasada $100\,mseg$, que es la que se suma a la muestra actual en cada interrupción.

Otro ajuste importante para el efecto Eco es la ganancia G de la señal retrasada sumada a la original, ya que debido a que se está tratando emular el efecto Eco producido naturalmente es necesario que la señal retrasada no tenga el mismo nivel de volumen que la original, ya que en el Eco natural la señal de audio original rebota en alguna superficie lo que produce el retraso y también una atenuación. Esta ganancia entonces no puede valer 1, lo que significaría que está al mismo nivel que la original, además provoca una mala distinción del mensaje original, por lo que al hacer pruebas se estableció la ganancia de la señal retrasada como la mitad de la original, es decir:

$$G_{SR} = 0.5 \qquad (4.4)$$

Donde G_{SR} es la ganancia de la señal retrasada.

La implementación final del efecto Eco es mostrada en la *figura 4.9*.

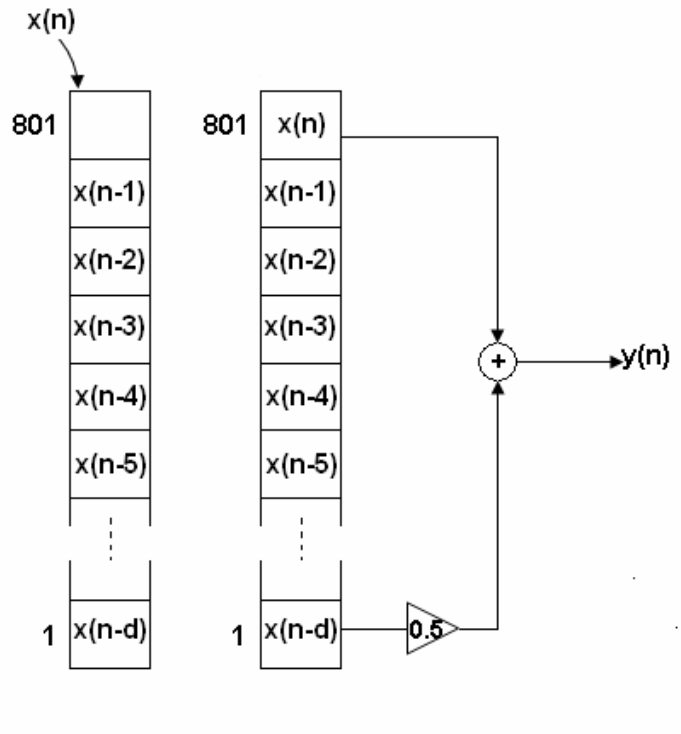

Figura 4.9 Efecto Eco.

El código final de la función "*eco*" es:

```
short eco(short nvalor)                  // Definición de la función eco.
  {
   int i;                                // Definición de variable entera.
   short res=0;                          // Definición de variable tipo short.
   buff[N+1]=nvalor;                     // Coloca muestra actual al final del buffer.
   res=data+0.5*buff[1];                 // Suma muestra actual con la muestra retrasada.

   for (i=N; i>-1; i--)                  // Barrido del buffer de memoria.
          {
          buff[N-i]=buff[N-i+1];         // Corrimiento de la muestra un lugar en el buffer.
          }
    return (short) res;                  // Regresa el resultado.
   }
```

Reverb.

El efecto denominado Reverb o Reverberación es fácil de implementar una vez funcionando el Eco, este efecto se implementa generando más de un efecto de Eco en la señal de salida, esto es:

$$y(n) = x(n) + G_1(x - d_1) + G_2(n - d_2) \qquad (4.5)$$

Al igual que en el efecto de Eco se definió un buffer para la señal de entrada, sin embargo en esta ocasión el vector se definió del doble de longitud que el definido para el efecto Eco, considerando tener dos Ecos, uno con el mismo retraso que el del efecto Eco y otro con el doble, es decir 100 *mseg* , o sea 801 muestras como se muestra en al *figura 4.10*.

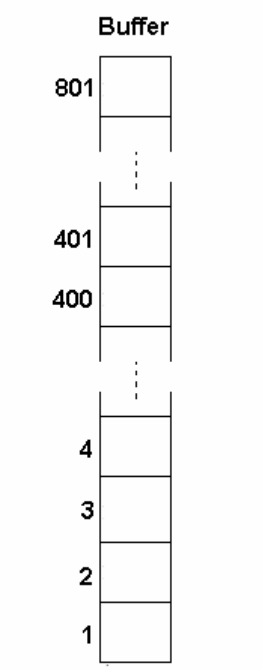

Figura 4.10 Buffer para el Efecto Reverb.

Cada vez que es llamada la función definida en este efecto, guarda la muestra actual al final del buffer de retraso, realiza el efecto Reverb al sumar la muestra actual con las muestras que se encuentran retrasadas en d_1 y d_2, que en este caso son las de las localidades 401 y 1 respectivamente, realiza el corrimiento en el buffer, y regresa el valor de la señal con Reverb, la *figura 4.11* muestra la suma de las muestras retrasadas y la muestra actual.

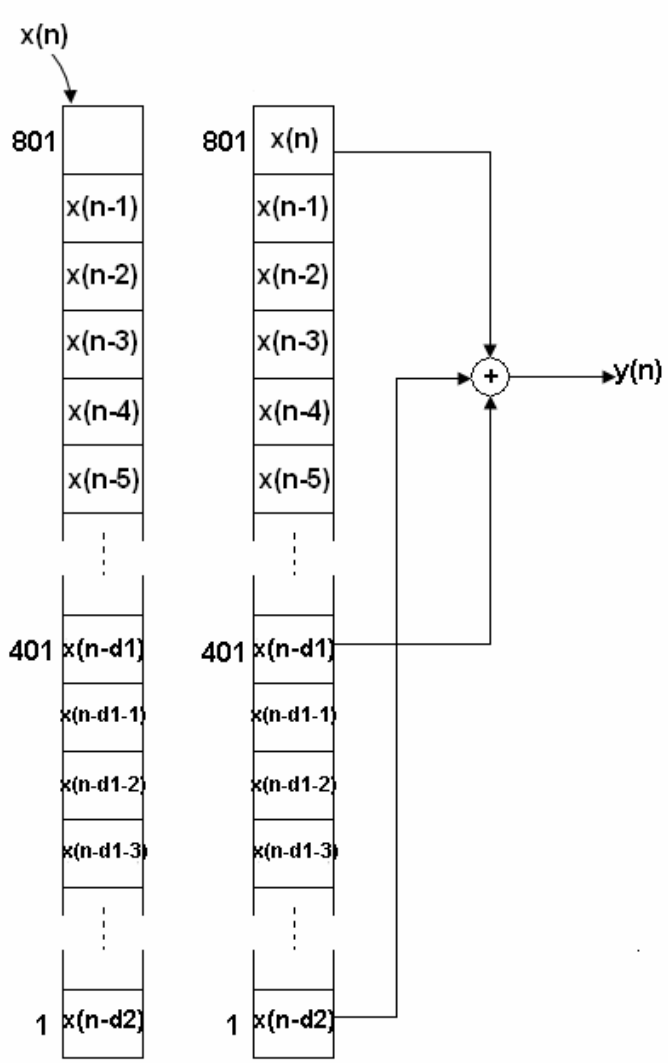

Figura 4.11 Generación del Efecto Reverb a partir del Buffer de Memoria

La función "*reverb*" también realiza un corrimiento de las muestras almacenadas en el buffer una vez realizada la suma de las muestras retrasadas con la muestra actual, dicho corrimiento se realiza mediante el siguiente código:

```
for (i=Nr*2; i>-1; i--)                    // Barrido del Buffer de memoria.
          {
          buff3[Nr*2-i]=buff3[Nr*2-i+1]; // Corrimiento de muestra un lugar en el buffer.
           }
```

Donde *Nr*2* es la longitud del buffer para el efecto Reverb.

La *figura 4.12* muestra el corrimiento realizado en el buffer por la función "*reverb*".

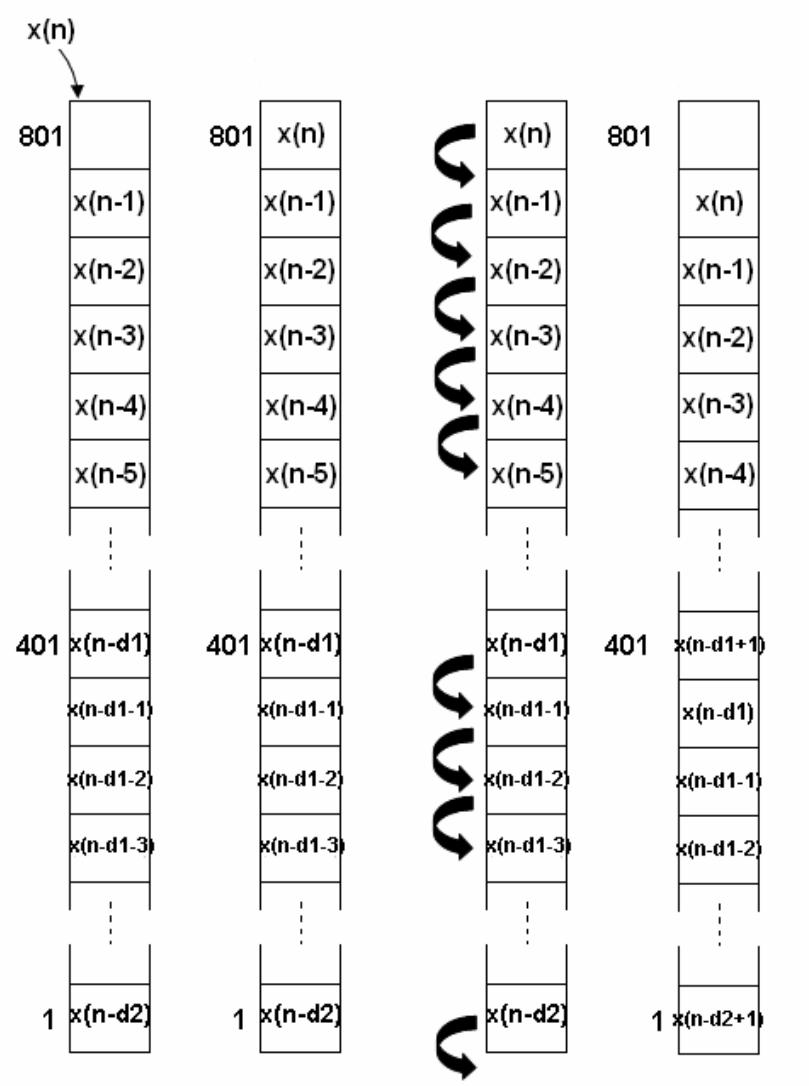

Figura 4.12 Corrimiento de Muestras en Memoria en la Función "*reverb*".

Ajustes al Efecto Reverb.

De los ajustes realizados al Eco se tienen los limites en el valor del retraso, que aplicados al efecto Reverb resultan en que el primer retraso no debe ser menor a $50\,mseg$ y el segundo no debe ser mayor a $200\,mseg$. Haciendo pruebas y manteniendo el valor del segundo retraso como el doble del primero, los retrasos adecuados son de $75\,mseg$ y $150\,mseg$ respectivamente, en número de muestras:

$$d_1 = (75mseg)\left(8000\frac{muestras}{seg}\right) = 600\,muestras \tag{4.6}$$

$$d_2 = (150 mseg)\left(8000 \frac{muestras}{seg}\right) = 1200\, muestras \tag{4.7}$$

El ajuste al efecto Reverb es definir un buffer para la señal de entrada de 1200 localidades de longitud, teniendo en cuenta que en la localidad 1 se encontrará la muestra retrasada $150\, mseg$ y en la localidad 601 la muestra retrasada $75\, mseg$, ambas se sumaran a la muestra actual cada que se presente la interrupción.

Al igual que en el efecto anterior se tiene que definir la atenuación de las dos señales retrasadas, emulando el efecto Reverb natural, para tener la sensación que la señal rebota en superficies diferentes lo que genera diferentes retrasos y diferentes atenuaciones, considerando que entre mas lejos esté la superficie el retardo es mayor así como la atenuación. Debido a lo anterior la ganancia de la señal con el retraso menor debe ser mayor que la del retraso más grande. Después de las pruebas para una buena audición del efecto se establecieron las ganancias de las señales que son:

$$G_{SR1} = 0.6 \tag{4.8}$$

$$G_{SR2} = 0.4 \tag{4.9}$$

Se observa entonces que los valores corresponden con los valores de los retrasos, teniendo una ganancia mayor para un retraso menor y una ganancia menor para un retraso mayor, incluso comparándolas con la ganancia en el retraso del efecto Eco.

La *figura 4.13* muestra la implementación final del efecto Reverb.

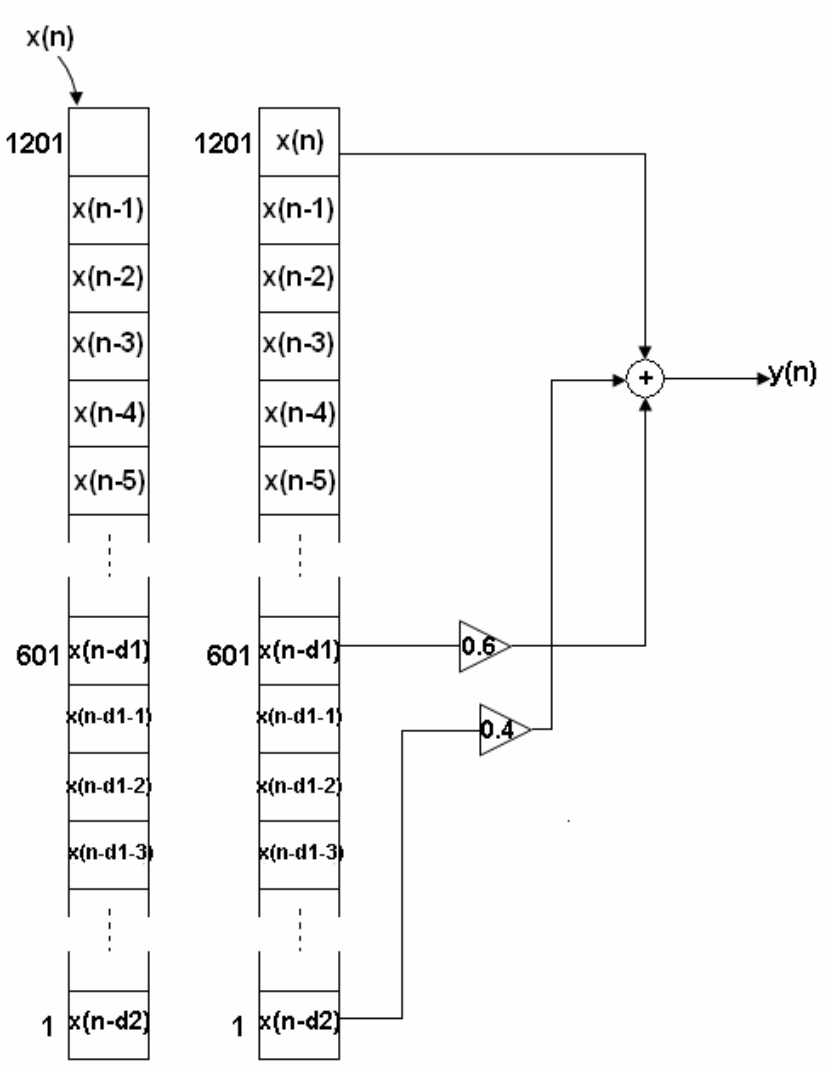

Figura 4.13 Efecto Reverb.

El código final de la función Reverb es:

```
short reverb(short nvalor)                    // Definición de la función reverb.
  {
   int i;                                     // Definición de variable entera.
   short res=0;                               // Definición de variable tipo short.
   buff3[Nr*2+1]=nvalor;                      // Coloca muestra actual al final del buffer.
   res=data+0.6*buff3[Nr+1]+0.4*buff3[1];     // Suma muestra actual con las muestras
                                              // Retrasadas.
   for (i=Nr*2; i>-1; i--)                    // Barrido del buffer de memoria.
          {
          buff3[Nr*2-i]=buff3[Nr*2-i+1]; // Corrimiento de la muestra un lugar en el buffer.
          }
    return (short) res;                       // Regresa resultado.
   }
```

Flanger.

Este efecto es uno de los llamados Ecos "modulados", ya que el retraso es variable de acuerdo a una señal senoidal:

$$y(n) = x(n) + gx[n - M(n)] \quad donde: \quad M(n) = [1 + Asen(2\pi fnT)] \tag{4.10}$$

Al igual que los efectos anteriores el efecto fue implementado en Matlab, debido a que se cuenta con todas las muestras de la señal de audio desde un principio, se implementó una función de redondeo al entero más próximo, y la función que variaba el valor del retraso.
Se puede observar en la *ecuación (4.10)* que la variación del retraso es en ambos sentidos, es decir el retraso puede ser tanto positivo como negativo, es decir un adelanto de la señal, lo que significaría que en algún momento la muestra que se sumaría a la muestra actual debe ser la muestra siguiente, lo cual es imposible en este caso ya que se implementa el efecto en tiempo real. El retraso en ambos sentidos se observa en la *figura 4.14*.

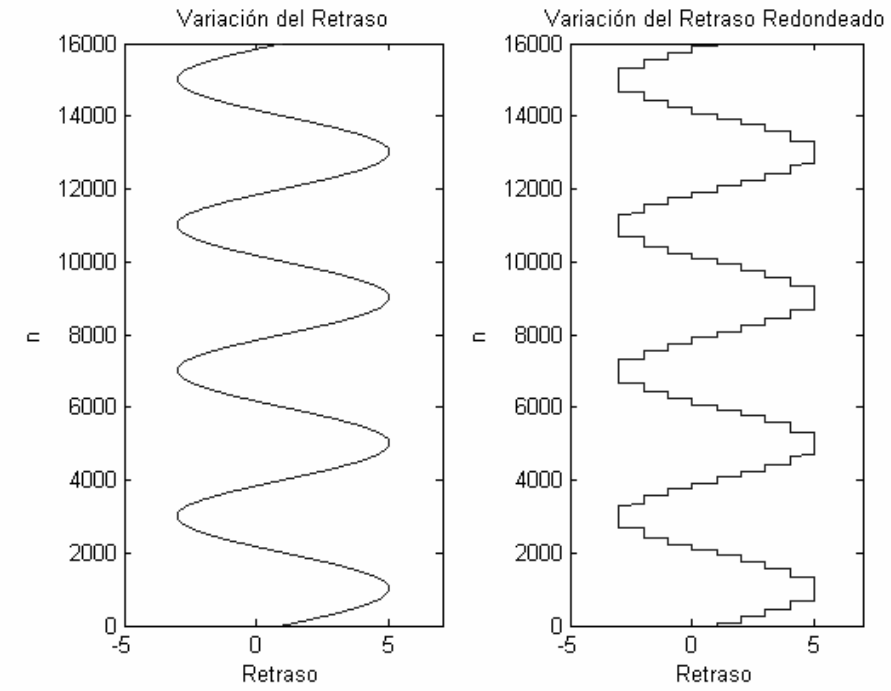

Figura 4.14 Retraso en la Función Flanger.

Debido a la no realización física de adelantos fue necesario modificar la función que varía el retraso para que solo produjera valores positivos, los cuales indican que muestra anterior se sumará a la actual, esto es:

$$y(n) = x(n) + gx[n - M(n)] \quad donde: \quad M(n) = [1 + A(1 + sen(2\pi fnT))] \tag{4.11}$$

Esta modificación no afecta el desempeño del efecto Flanger ya que el retraso solo presenta un límite en su variación, es decir solo presenta números positivos, como se ve en la *figura 4.15*.

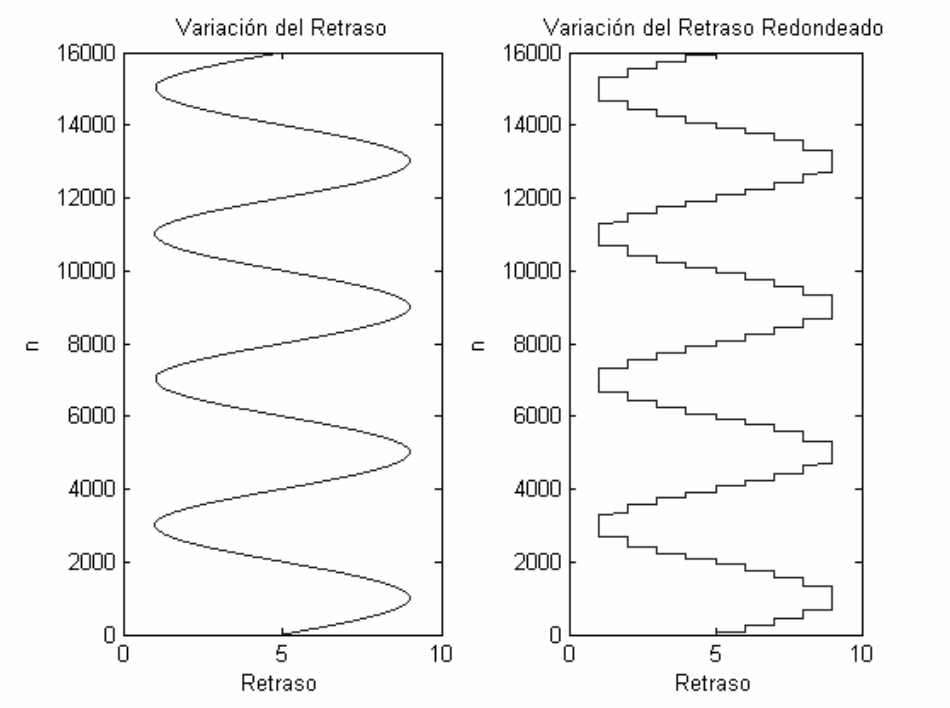

Figura 4.15 Retraso en la Función Flanger en Tiempo Real.

Al igual que en los efectos anteriores, se definió un buffer de la señal de entrada, de menor longitud que en los efectos anteriores, ya que el retraso que se utiliza en el Flanger es menor que el del Eco, aproximadamente de $10\,mseg$, y depende de que tanto se le deje variar en la *ecuación (4.5)*. La longitud del buffer se controla con una variable *N*, definida al principio del programa, la cual también se incluye en la ecuación implementada para el Flanger, con el fin de variar el intervalo del retraso al mismo tiempo que se varía el tamaño del buffer de la señal, la ecuación implementada en el DSP es:

$$y(n) = x(n) + buff\left[1 + retraso \times \left(N/10\right)\right] \quad donde: \quad retraso = 1 + 4(1 + sen(2\pi fnT)) \qquad (4.12)$$

De la *ecuación (4.12)* se observa que tanto el tamaño del buffer como el máximo retraso varían proporcionalmente a las variaciones de *N*. Además de la *ecuación (4.12)* el máximo valor que puede tomar la variable *retraso* es 9, por lo que en la ecuación la variable *N* es divida entre 10, así con un solo barrido de valores de la variable *retraso* entre 1 y 9 se abarca el tamaño del buffer, que en un principio se definió de 81 localidades, es decir $N = 81$, como se observa en la *figura 4.16.*

Para realizar este efecto se implementó la función "*round*" que realiza el redondeo, se definió la función "*delay*" y "*flanger*", esta última es llamada cada interrupción, guarda la muestra actual en el buffer, manda llamar la función "*delay*" que es la encargada de dar el valor del retraso de acuerdo a la *figura 4.15*, en este caso el número de muestra que se suma a la muestra presente, la función "*delay*" a su vez llama a la función "*round*" para entregar un valor entero ya que no se puede buscar una localidad fraccionaria en el buffer. La función

"*flanger*" suma la muestra actual con la muestra guardada en la localidad del buffer que indique el retraso, y por último realiza un corrimiento de las muestras en el buffer.

La función "*round*" esta definida por el siguiente código:

```
short round(short valor)                    // Definición de la función round.
    {
     int j;.                                // Definición de variable entera.
     short b=0;                             // Definición de variable de tipo short.
     short rou=0;                           // Definición de variable de tipo short.
     for (j=0; j<10; j++).                  // Ciclo de barrido de 0 a 10.
     {
       b=valor-j;                           // b = valor a redondear – j.
       if (b<0.5)                           // Sí b es menor a 0.5.
       { rou=j;.                            // Valor redondeado = j.
         break;}                            // Detiene el ciclo.
       if (b<1)                             // Sí b es mayor a 0.5.
       { rou=j+1;.                          // Valor redondeado = j+1.
         break;}                            // Detiene el ciclo.
     }
     return (short) rou;                    // Regresa resultado.
    }
```

La función "*delay*" esta definida por el siguiente código:

```
short delay(void).                          // Definición de la función delay.
   {
    short m=0;                              // Definición de variable tipo short.
    if (n==16000)                           // Sí n es igual a 16000.
     {
      n=0;                                  // Reinicia variable n.
     }

    m=(1+4*(1+sin(2*pi*f*(n++)/8000)));     // Calcula retraso.
    m=round(m);                             // Redondea resultado.
    return (short) m;                       // Regresa resultado.
    }
```

La *figura 4.16* muestra la generación del efecto Flanger a partir de las muestras guardadas en el buffer y de la variación del retraso.

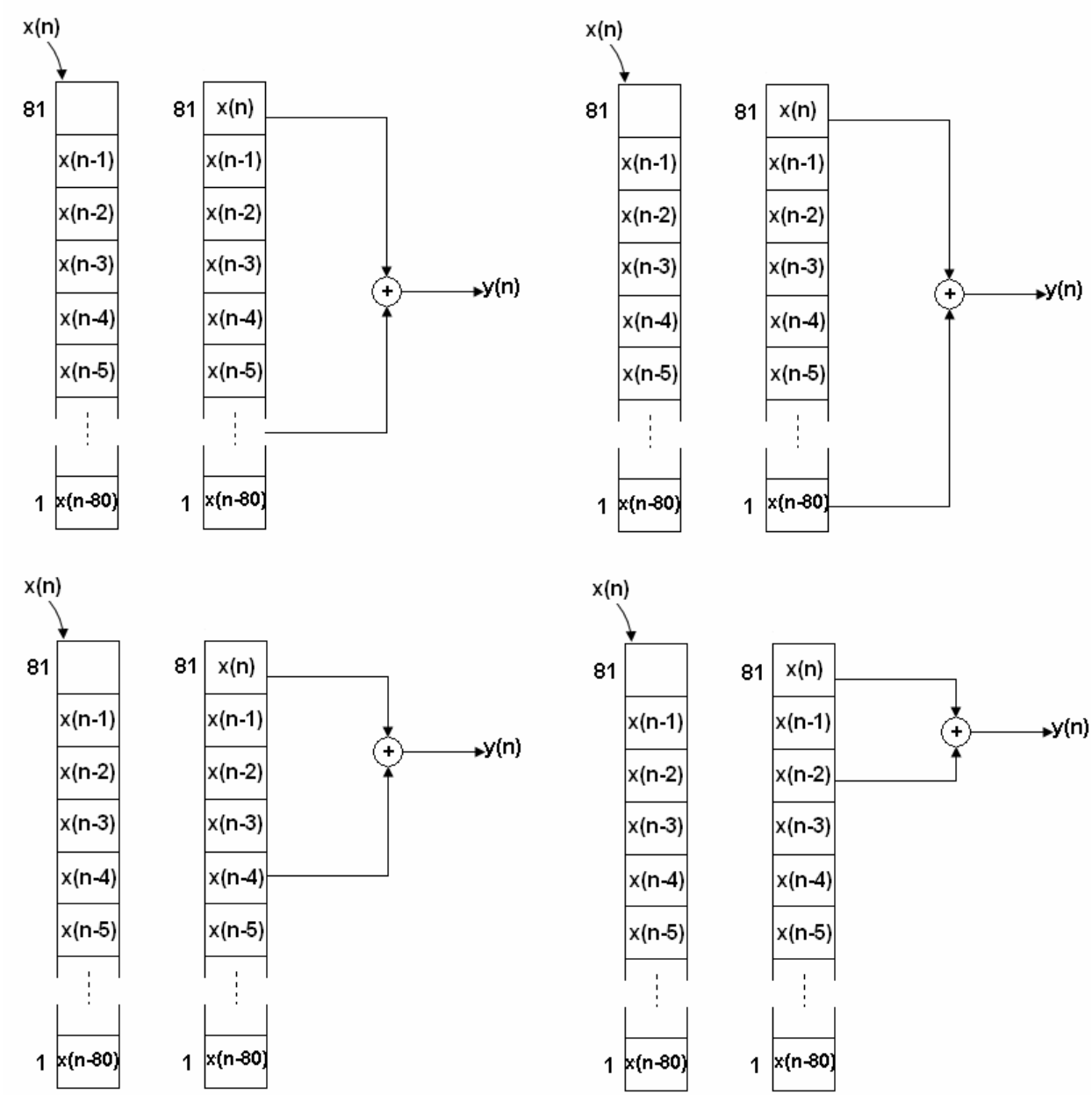

Figura 4.16 Generación del Efecto Flanger a partir del Buffer de Memoria.

Observando la *figura 4.16* se nota la ausencia del valor d que representa el retraso, esto es debido a que el retraso es variable y no tiene un valor fijo que se pueda definir para cualquier instante.

En la *figura 4.17* se presenta el corrimiento en el buffer de memoria realizado por la función "*flanger*".

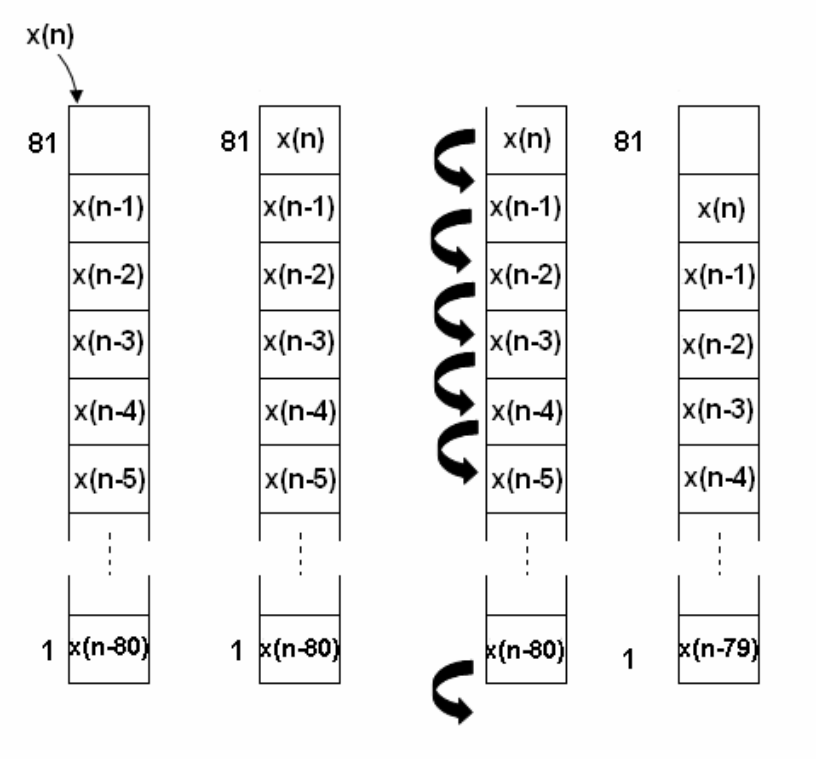

Figura 4.17 Corrimiento de Muestras en Memoria en la Función "*flanger*".

Observando la ecuación del Flanger (*ecuación (4.11))* notamos la presencia de un parámetro fundamental para el desempeño de este efecto que es la frecuencia de modulación del retraso, la cual nos indica que tan rápido varía este, para que el efecto Flanger se aprecie el retraso debe variar cada 2 segundos aproximadamente, lo que implica una frecuencia de modulación de 0.5 Hz.

Ajustes al Efecto Flanger.

De todos los efectos implementados el Flanger es el que tiene mas parámetros en su ecuación fundamental, generando un número importante de combinaciones de valores de dichos parámetros que generan a su vez un igual número de posibles desempeños de este efecto. Se debe tener cuidado de que el retraso no sea muy grande, es decir, que no sea audible porque si no se tendría un efecto de Eco y se pierde completamente el efecto Flanger.
La frecuencia de modulación del retraso también debe tener un valor adecuado ya que si es demasiado grande o demasiado pequeña, el efecto Flanger está presente, sin embargo, no es apreciable al oído humano y se tiene una señal de salida sin efecto especial audible alguno. Por lo que siguiendo la teoría, la frecuencia de modulación se estableció en $0.5\,Hz$, para realizar pruebas modificando los demás parámetros.

El parámetro A, que controla la oscilación máxima del retraso, en la *ecuación (4.11)* se dejó con valor de 4 que es el valor empleado en el programa en Matlab, ya que en las primeras pruebas realizadas en tiempo real se percibió que este valor permitía un funcionamiento

adecuado además de que con este valor se puede asociar el tamaño del buffer con el retraso máximo que tendrá el efecto.

En la *ecuación (4.11)* también se observa un parámetro T que se refiere al periodo de muestreo que en este caso es:

$$T = \frac{1}{f_S} = \frac{1}{8000\, seg^{-1}} = 0.125\, mseg \tag{4.13}$$

Al observar la ecuación implementada en el DSP (*ecuación (4.12)*) para el efecto Flanger se pueden conocer los valores máximo y mínimo del retraso en función del tamaño del buffer definido por la variable *N*, que son:

$$R_{mínimo} = \frac{N}{10} + 1\, muestras \tag{4.14}$$

$$R_{máximo} = 9\left(\frac{N}{10}\right) + 1\, muestras \tag{4.15}$$

El retraso en milisegundos está dado por:

$$R_{mínimo} = \left(\frac{N}{10} + 1\right)\left(\frac{1}{f_S}\right) = \frac{\left(\frac{N}{10}\right)+1}{8000}\, mseg \tag{4.16}$$

$$R_{máximo} = \left(9\left(\frac{N}{10}\right) + 1\right)\left(\frac{1}{f_S}\right) = \frac{9\left(\frac{N}{10}\right)+1}{8000}\, mseg \tag{4.17}$$

Realizando pruebas y variando el retraso máximo, en este caso modificando el valor de la variable *N*, se obtuvo el valor adecuado para este efecto que es $N = 100$.

Por lo que los retrasos máximo y mínimo son:

$$R_{mínimo} = \frac{\left(\frac{100}{10}\right)+1}{8000} = \frac{11}{8000} = 1.4\, mseg \tag{4.18}$$

$$R_{máximo} = \frac{9\left(\frac{100}{10}\right)+1}{8000} = \frac{91}{8000} = 11.4\, mseg \tag{4.19}$$

El valor del retraso mayor asegura que no se oirá Eco en la señal de salida, y por lo tanto el efecto Flanger funcionará correctamente. La *figura 4.18* presenta el valor del retraso en muestras del efecto Flanger final.

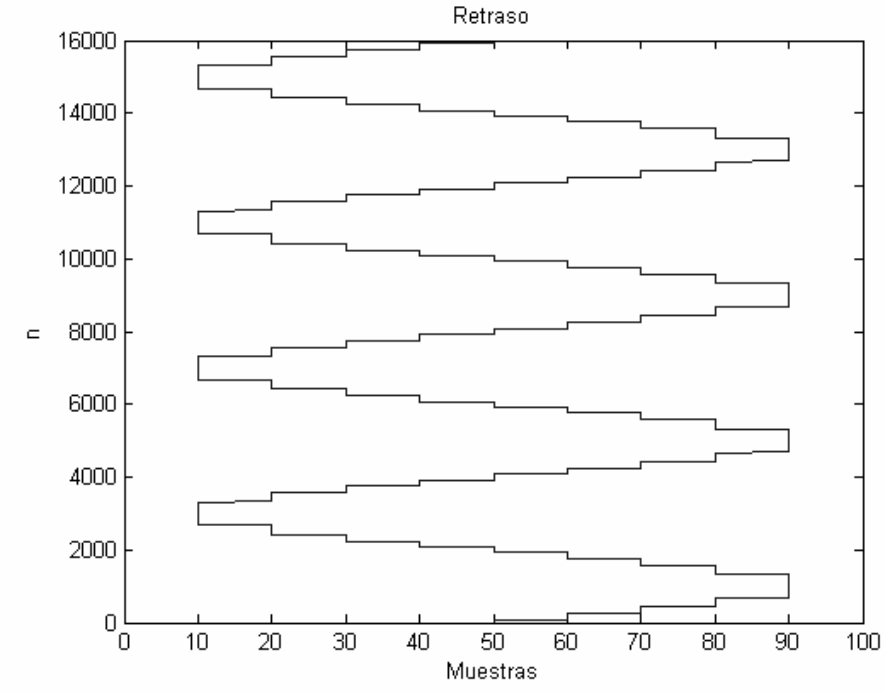

Figura 4.18 Variación del Retraso en el Efecto Flanger.

La *figura 4.19* muestra la implementación final del efecto Flanger.

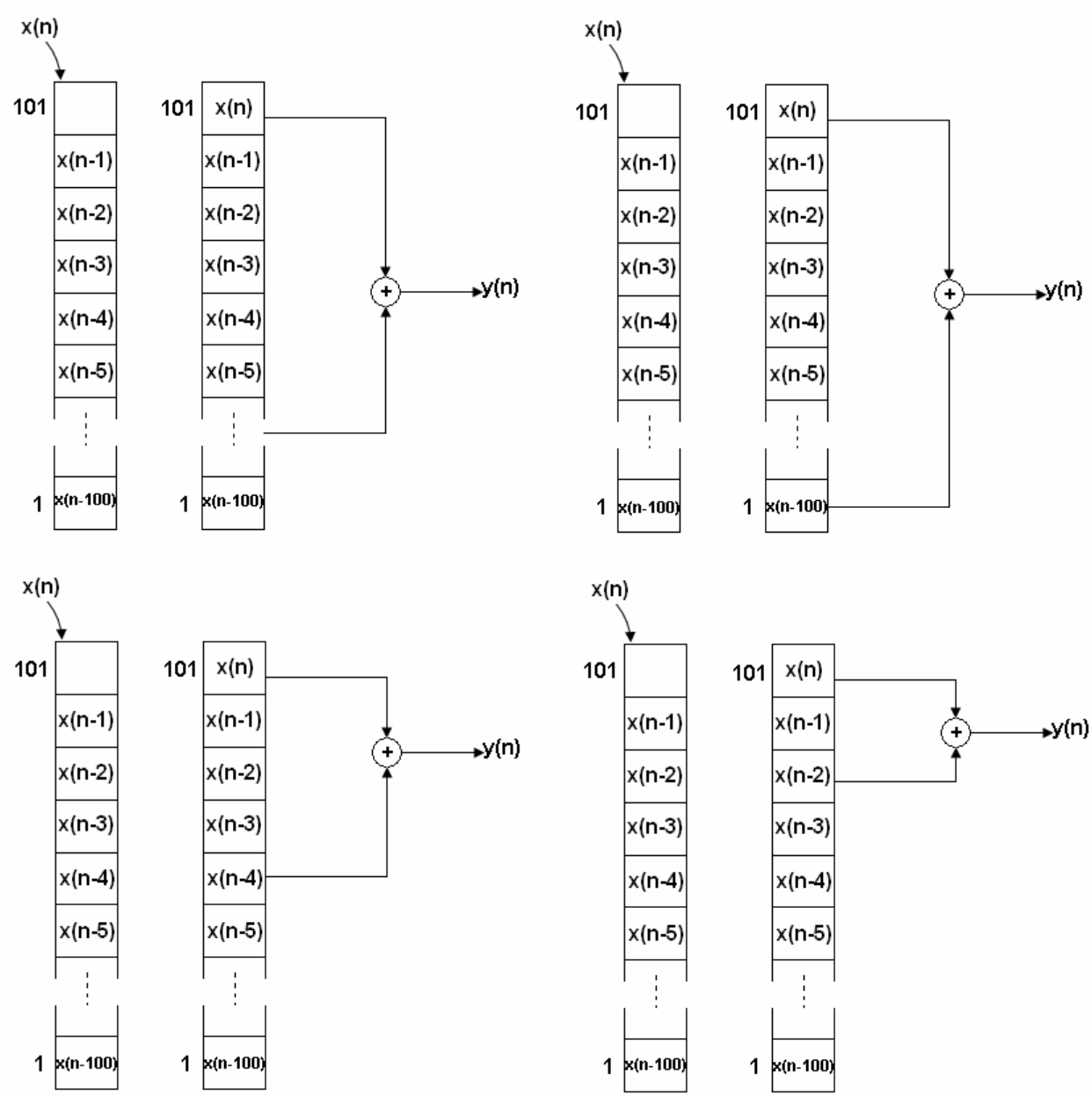

Figura 4.19 Flujo de Muestras en Memoria para el Efecto Flanger.

El código final de las funciones "*round*", "*delay*" y "*flanger*" son:

```
/*Función para redondear*/

short round(short valor)                    // Definición de la función round.
    {
     int j;.                                // Definición de variable entera.
     short b=0;                             // Definición de variable de tipo short.
     short rou=0;                           // Definición de variable de tipo short.
     for (j=0; j<10; j++).                  // Ciclo de barrido de 0 a 10.
     {
```

```
       b=valor-j;                                  // b = valor a redondear – j.
       if (b<0.5)                                  // Sí b es menor a 0.5.
       { rou=j;                                    // Valor redondeado = j.
         break;}                                   // Detiene el ciclo.
       if (b<1)                                    // Sí b es mayor a 0.5.
       { rou=j+1;                                  // Valor redondeado = j+1.
         break;}                                   // Detiene el ciclo.
     }
     return (short) rou;                           // Regresa resultado.
    }

/*Función varia el retraso*/

short delay(void)                                  // Definición de la función delay.
   {
    short m=0;                                     // Definición de variable tipo short.
    if (n==16000)                                  // Sí n es igual a 16000.
     {
      n=0;                                         // Reinicia variable n.
     }

    m=(1+4*(1+sin(2*pi*f*(n++)/8000)));            // Calcula retraso.
    m=round(m);                                    // Redondea resultado.
    return (short) m;                              // Regresa resultado.
    }

/*Función que realiza el Flanger*/

short flanger(short nvalor)                // Definición de la función flanger.
  {
   int i;                                  // Definición de variable entera.
   short res=0;                            // Definición de variable tipo short.
   retraso=delay();                        // Llama función del retraso.
   buff2[Nu+1]=nvalor;                     // Coloca muestra actual al final del buffer.

   res=data+G*buff2[1+(retraso*(Nu/10))];// Suma muestra actual con la muestra retrasada.
   for (i=Nu; i>-1; i--)                   // Barrido del buffer de memoria.
          {
          buff2[Nu-i]=buff2[Nu-i+1];       // Corrimiento de la muestra un lugar en el buffer.
          }
    return (short) res;                    // Regresa el resultado.
   }
```

Silenciar.

Este efecto al ser implementado en Matlab genera una señal de ganancia la cual tiene solo dos valores posibles 0 y 1, que se multiplica por la señal de entrada y así dejar audible solo partes de la señal.

El único cambio en este efecto es de algoritmo que genere la señal de ganancia en tiempo real. Esto se implementó a partir de una señal senoidal, generando la señal de ganancia de valor 1

cuando la señal senoidal es positiva y valor 0 cuando la señal senoidal es negativa como se muestra en la *figura 4.20.*

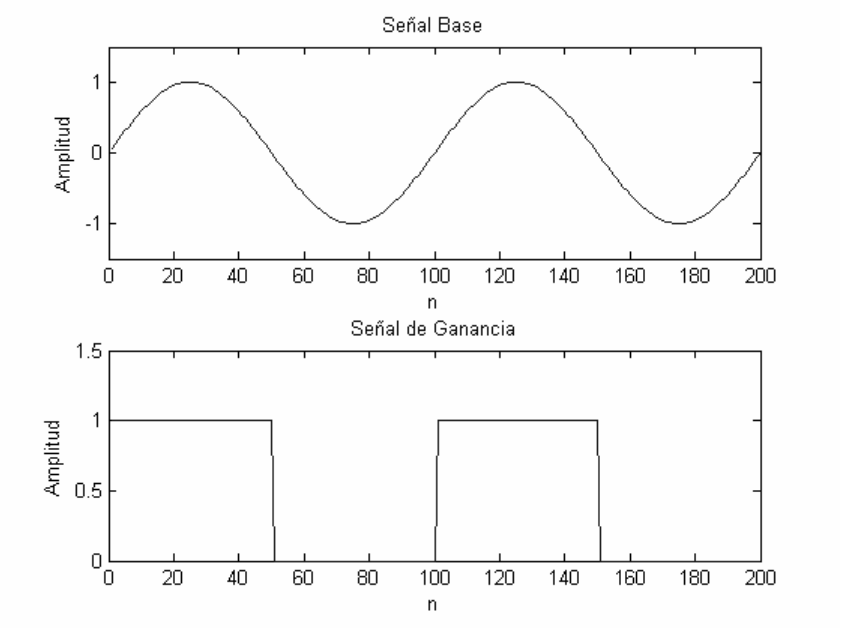

Figura 4.20 Señal de Ganancia en el Efecto Silenciar.

El algoritmo que genera la señal de ganancia se implementó con el siguiente código:

```
c=sin(2*pi*fc*(n++)/8000);          // Calcula señal de ganancia base.

    if (c>0)                        // Sí el valor de la señal base es mayor a 0.
     {
       g=1;                         // Señal de ganancia igual a 1.
      }
    else                            // Caso contrario.
     {
       g=0;                         // Señal de ganancia igual a 0.
     }
```

Se implementó la función "*silencio*", la cual genera la señal de ganancia cada vez que es llamada en una interrupción, utilizando una frecuencia definida por N_S que es la suma del número de muestras que se desean dejar audibles y el número de muestras que se eliminarán de la señal de salida , esto es:

$$fc = \frac{fs}{N_S} \tag{4.20}$$

Donde f_C es la frecuencia de la señal de ganancia y f_S es la frecuencia de muestreo.

La función multiplica la muestra actual por el valor de la señal de ganancia en ese instante, dejando la muestra actual audible o no, dependiendo del valor de la ganancia.

Ajustes al Efecto Silenciar.

Este efecto no tiene mas que un que parámetro para modificar su funcionamiento, dicho parámetro es la frecuencia de la señal de ganancia, sin embargo, en la ecuación que define dicha frecuencia (*ecuación (4.20)*), se observa que varía directamente proporcional a la frecuencia de muestreo (f_S), e inversamente proporcional a la suma del número de muestras que se desean silenciar de la señal de entrada y el número de muestras que se dejan audibles, esto es N_S , como la frecuencia de muestreo es fija para todos los efectos, el ajuste para este efecto se basa en variar el número de muestras audibles y no audibles.

Este efecto es el mas fácil de implementar para una señal de audio de una longitud dada, a la cual se le quita o se silencia una parte, sin embargo, en la implementación en tiempo real se van silenciando varias partes de la señal conforme varía la señal de ganancia, lo que implica un funcionamiento distinto, al implementar la función de ganancia no se tiene un intervalo de valores ideal para su frecuencia, lo que implica que el ajuste tiene que realizarse variando el número de muestras hasta que se obtenga el efecto deseado.

Al realizar las pruebas el ajuste adecuado se encontró para un número de muestras de $N_S = 1500$, por lo que la frecuencia de la señal de ganancia es:

$$f_C = \frac{8000 \frac{muestras}{seg}}{1500 muestras} = 5.333 seg^{-1} = 5.333 Hz \qquad (4.21)$$

Además al probar el efecto se notó que la señal con el efecto Silenciar se escuchaba menos comparada con las señales que presentaban los demás efectos, así que se cambio el valor de la señal de ganancia a 2.

El código final de la función "*silencio*" es:

```
short silencio(short nvalor)          // Definición de la función silencio.
  {
    short res=0;                      // Definición de variable tipo short.

    fc=(fs/Ns);                       // Frecuencia señal de ganancia.
    if(n==Ns)                         // Si n es igual a Ns.
     {
      n=0;                            // Reinicia variable n.
     }

c=sin(2*pi*fc*(n++)/8000);            // Calcula señal de ganancia base.

    if (c>0)                          // Sí el valor de la señal base es mayor a 0.
     {
       g=2;                           // Señal de ganancia igual a 2.
      }
    else                              // Caso contrario.
```

```
    {
      g=0;                          // Señal de ganancia igual a 0.
    }

   res=g*nvalor;                    // Multiplica muestra actual por la señal de ganancia.

   return (short) res;              // Regresa el resultado.
  }
```

4.3 Integración del Sistema de Efectos Especiales de Audio en Tiempo Real.

Una vez probados los efectos, funcionando individualmente en tiempo real y ajustado sus parámetros, se procedió a integrarlos en un solo proyecto, es decir, un sistema en el cual se pueda seleccionar en tiempo real uno de los efectos especiales de audio, es decir, no importa en que momento se realice el cambio de efecto, la señal de salida debe presentar el efecto seleccionado, además no debe de haber interrupción alguna en el audio que se escucha como resultado del sistema.

Los efectos implementados en le DSP fueron desarrollados pensando que solo se tenía uno por proyecto, sin embargo, ahora se tienen que ajustar para que coexistan en un solo proyecto, además de que no exista interferencia entre ellos.

4.3.1 Rutina de Selección de los Efectos de Audio.

La parte fundamental del sistema completo es el cómo hacer para que los efectos se puedan seleccionar entre sí, y mas aun que se puedan seleccionar en cualquier momento que el usuario desee. Para su realización es necesario primero hacer una rutina de selección e integrar el sistema. La *figura 4.21* muestra un diagrama a bloques general del sistema.

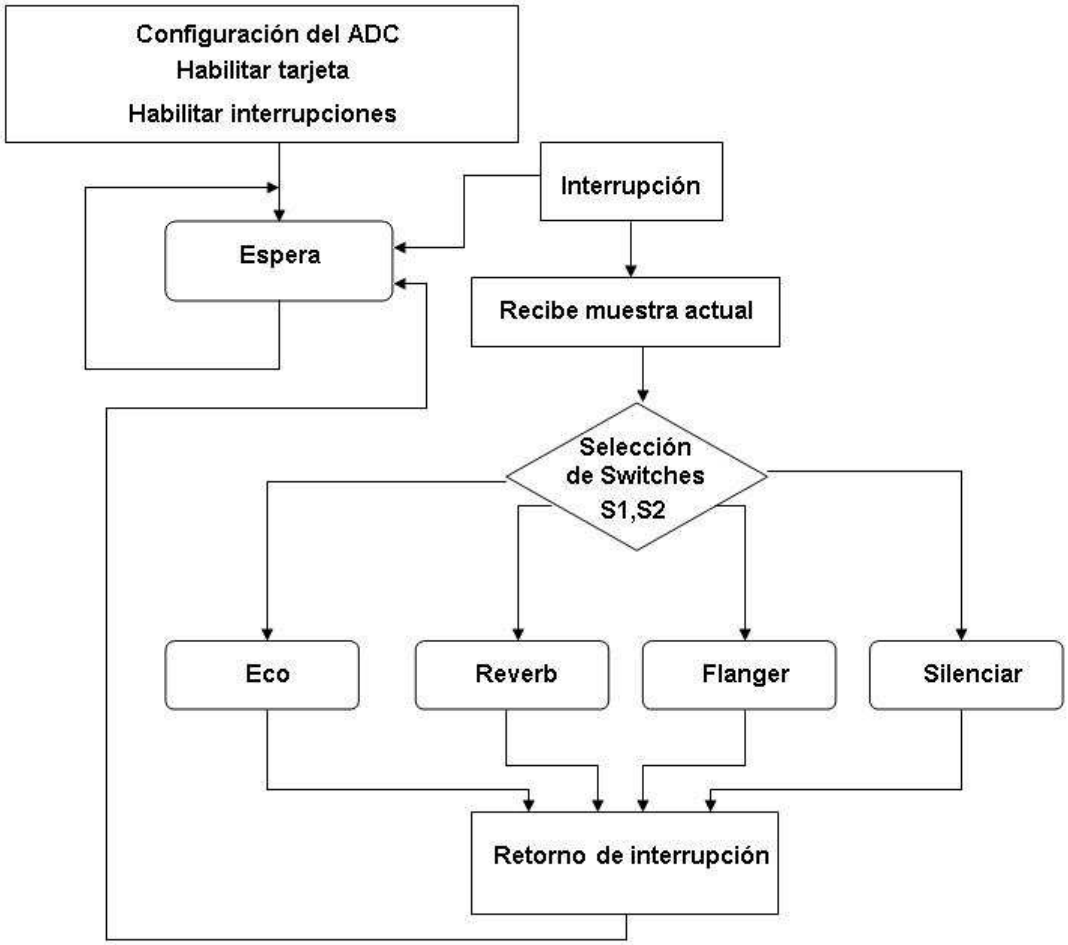

Figura 4.21 Diagrama de Flujo del Sistema.

La rutina de selección se implementó a partir del proyecto original, el cual fue base para todos los efectos especiales.

La idea principal de la selección de los efectos de audio es usar los dos interruptores de usuario con los que cuenta la tarjeta de desarrollo del DSP, teniendo cuatro combinaciones posibles que permiten elegir igual número de efectos.

El estado de los interruptores, es decir si están cerrados o abiertos, se ve reflejado en el valor que presenta el puerto uno (*port1*) en la tarjeta de desarrollo, por lo que hay que obtener que valores toma dicho puerto cuando están presentes cada una de las combinaciones que el usuario tiene la posibilidad de establecer. Dichos valores en su forma decimal son 0, 32, 64, y 96, ya que no se debe alterar el estado de los otros bits del puerto (ver *figura 4.22*).

Para la selección hay que hacer una rutina que lea el valor del puerto uno, y con base a sentencias condicionales determine que efecto es el que se debe aplicar sobre la señal de audio de entrada. Las sentencias condicionales más adecuadas son "*switch*" y "*case*", ya que solo se necesita aplicar la sentencia "*switch*" y establecer los casos posibles con la opción "*case*", evitando varias comparaciones del mismo valor, como si se usara una sentencia "*if*".

Esta sentencia condicional se implementó como parte de la interrupción principal, esto con el fin de que cada vez que se presente una muestra de la señal de entrada, se tenga la decisión de que se debe hacer con esa muestra, teniendo así el control en tiempo real, y en cualquier instante que se desee, como se muestra en la *figura 4.21*.

La implementación final de la rutina de selección se hizo mediante el siguiente código:

```
short datoP1;
     data=*drr;
     datoP1 = port1;
     switch(datoP1)
      {
        case 0:
          data = reverb(data);
          break;
        case 32:
          data = eco(data);
          break;
        case 64:
          data = flanger(data);
          break;
        case 96:
          data = silencio(data);
          break;
      }
```

En la *figura 4.22* es mostrada la selección de los efectos según la posición del selector conectado al puerto *Port1*.

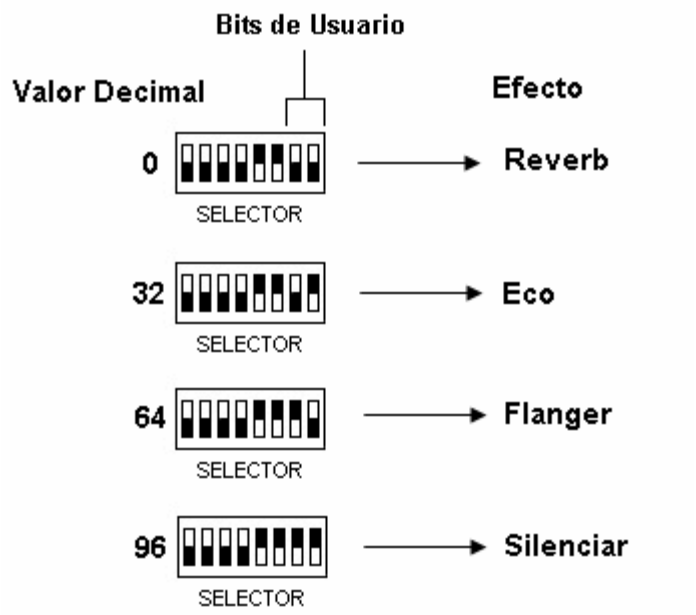

Figura 4.22 Selección de los Efectos a través de los Interruptores.

4.3.2 Creación del Proyecto.

Debido a la capacidad de procesamiento del DSP se pueden integrar todos los efectos considerados en un solo proyecto, que utilice todos los archivos comunes a los proyectos anteriores, e implementar un archivo "*.c*" conteniendo la información de cada uno de los archivos "*.c*" creados anteriormente.

Un problema que se tiene es que la mayoría de los efectos es que utilizan un buffer para la señal de entrada, y dichos buffers son de diferente tamaño dependiendo del efecto de audio, además de que cuando se implementaron los efectos individualmente, había una pequeña subrutina en la interrupción principal, la cual llenaba el buffer correspondiente muestra por muestra hasta que estuviera lleno y después el programa empezaba a realizar el efecto deseado.
Ahora bien al unir todos los efectos de audio en un solo proyecto no es posible tener esta subrutina para llenar el buffer ya que los tamaños son diferentes, la subrutina necesitaría que primero se realice la selección, y tener una sentencia condicional para decidir que buffer llenar, y cuando empezar a producir el efecto.
Analizando la integración de todos los efectos se observa que si se llena primero el buffer y después se comienza a realizar el efecto de audio, se presenta un retraso entre la señal original y la señal de salida, afectando el funcionamiento en tiempo real del sistema. Dicho retraso puede ser de un valor tal (aproximadamente $30\,mseg$) que no sea perceptible al oído humano, sin embargo, en el estricto sentido del funcionamiento en tiempo real es incorrecto.

Una vez eliminada la subrutina en la que se llena el buffer, la interrupción principal del programa solo debe constar de la rutina para la selección del efecto especial que se desea oír. Incluso el programa principal solo debe contener las instrucciones para inicializar la tarjeta de desarrollo, así como las instrucciones para definir los parámetros de la interrupción, teniendo solo un archivo fuente de extensión "*.c*".
Para tener un sistema con un esquema modular es necesario tener archivos donde estén definidas tanto las variables, como las funciones involucradas en el funcionamiento de los efectos especiales de audio. Se crearon dos archivos de extensión "*.h*", para definir variables y funciones, de ahí sus nombres "*variables.h*" y "*funciones.h*". El primero contiene la definición de las variables necesarias, como el nombre, tipo y tamaño, considerando que algunas variables en los proyectos individuales anteriores son del mismo tipo, así que en algunos de esos programas se repiten los nombres de las variables, sin embargo, al incluirlas en este archivo se les cambió el nombre para que todas las variables diferentes, tuvieran también nombres diferentes, las variables comunes y las constantes fijas utilizadas se definieron igual que en los proyectos anteriores.
Para el archivo "*funciones.h*" se incluyeron todas las funciones necesarias para producir los efectos especiales, con las respectivas modificaciones debido al cambio de nombre de algunas variables, y se indica de qué efecto es cada función así como lo que realiza cada una.

La *figura 4.23* muestra el conjunto de archivos que integran el sistema.

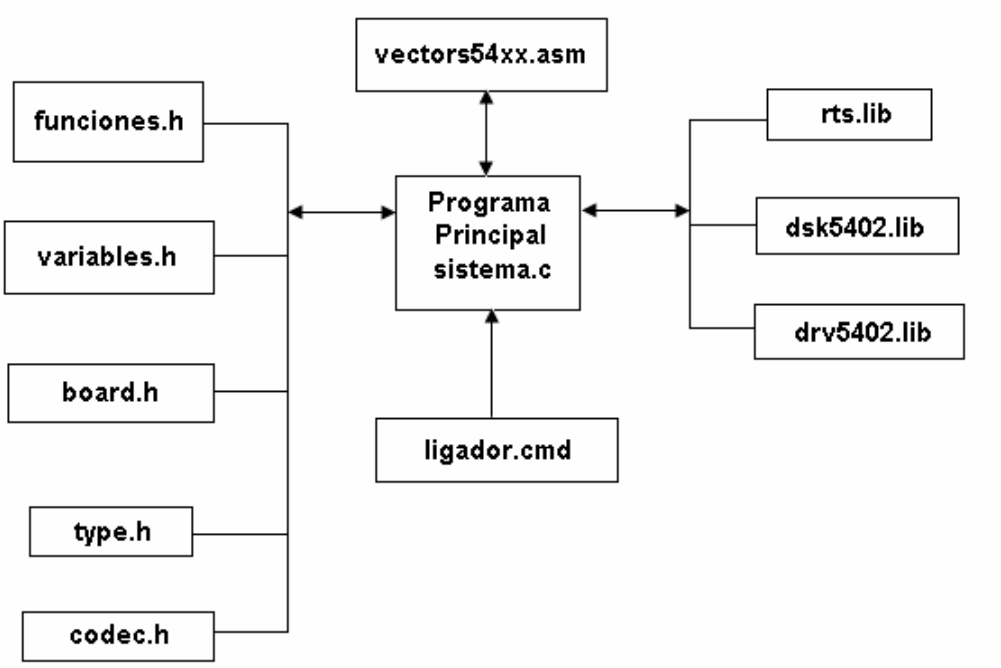

Figura 4.23 Archivos del Sistema.

Con todos los archivos mostrados en la *figura 4.23* se integra el proyecto, y utilizando las herramientas del CCS se genera el archivo "*sistema.out*" el cual se carga en el DSP para poderlo ejecutar. Una vez cargado el proyecto en el DSP se habilita la opción "*Run*" en el CCS para que el DSP funcione de acuerdo a las instrucciones contenidas en el archivo "*sistema.out*", lo cual permite evaluar el funcionamiento del sistema completo.

4.4 Grabación del Sistema en la Memoria Flash de la Tarjeta de Desarrollo.

Una vez teniendo el sistema integrado y probado se procedió a grabarlo en la memoria Flash de la tarjeta de desarrollo, para lograr la independencia del sistema de una PC.

Los DSPs TMS320VC5402 y TMS320UC5402 poseen un programa de arranque que reside en la memoria ROM interna del chip. La función de este programa es transferir código desde una fuente externa hasta la memoria de programa tan pronto como se inicializa el DSP. Esto permite que el código de una aplicación resida en una memoria externa no volátil y que una vez activado el DSP, dicho código sea transferido hacia al memoria programa. Para nuestro caso el programa de arranque trasladará el código de la aplicación de usuario desde la memoria Flash en el espacio de memoria dato hacia la DARAM en el espacio de memoria programa [2].

Existen al menos cinco métodos básicos de arranque del DSP (*figura 4.24*), mismos que se encuentran implementados dentro del código de programa de arranque que se encuentra alojado en la memoria ROM del chip C5402. Durante su operación el programa de arranque realiza una serie de pasos con la finalidad de determinar cual es el modo de inicialización a emplear. Para ello el programa de arranque verifica que se cumplan una serie de condiciones y

si estas son verdaderas, se escoge el método de inicio al que hacen referencia, si no se cumplen, entonces el programa operará hasta que se reúnan las condiciones para utilizar un método. Reunidas las condiciones para la elección de un método, el programa de arranque inicia la transferencia del código de acuerdo a las especificaciones proporcionadas por la tabla de inicio que el usuario generó a través de la utilería hex500. Finalizada la operación anterior el DSP estará listo para arrancar la ejecución de la aplicación a partir del punto de inicio. [2]. La *figura 4.24* muestra un esquema general del proceso de arranque.

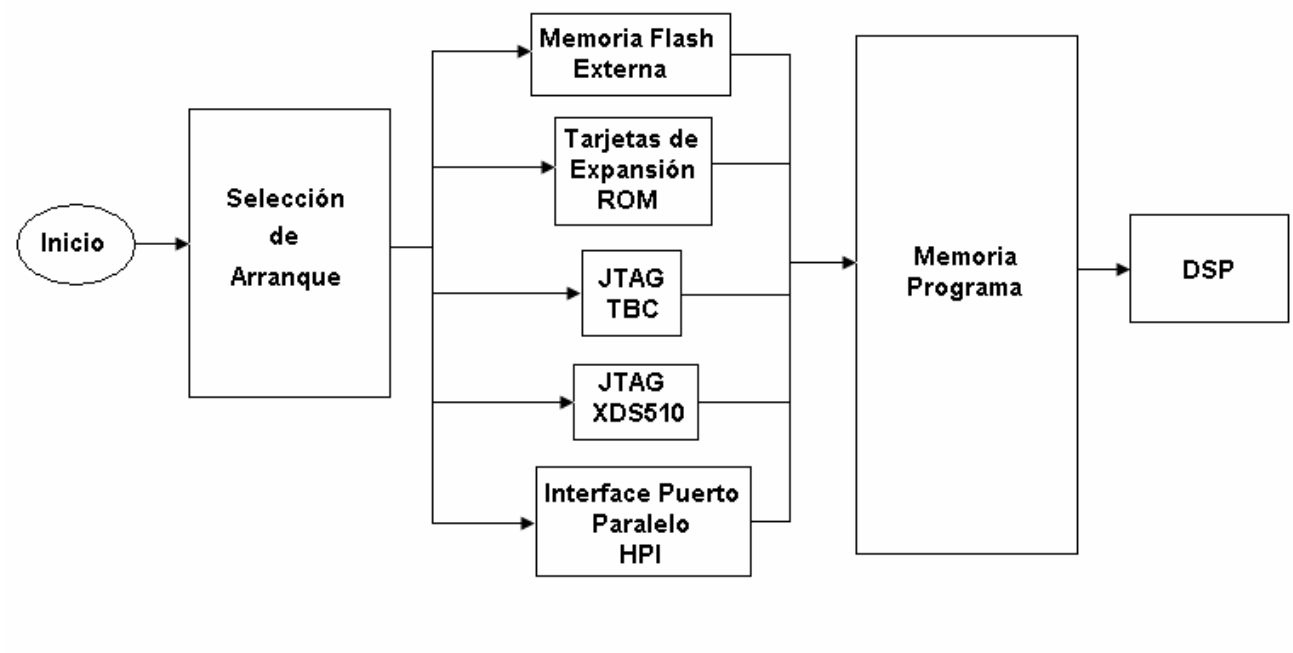

Figura 4.24 Procedimiento de Arranque del DSP.

Se siguió el procedimiento indicado en la nota de aplicación *TMS320VC5402 Flash Bootloader, Nota de Aplicación. Revisión B,* para la grabar la memoria Flash del DSP, el cual es el siguiente:

- **Alistar la tarjeta.** Establecer la configuración de la *Tabla 4.1* en los interruptores.

- **Verificar el mapa de memoria y la localización de las secciones.**

- **Eliminar errores en el programa.** Asegurarse de colocar las secciones de código y datos por debajo de la dirección 0x4000.

- **Generar un nuevo archivo de salida.** Construir de nuevo el proyecto incluyendo la opción **–V548** en el proceso de compilación, esta opción le indica al compilador y al ensamblador que se está trabajando con un DSP *C548*, a pesar de ser en realidad un *C5402*, esta opción solo habilita la utilidad *HEX500* para generar la tabla de arranque.

- **Crear el archivo con extensión "*.hex*".** a partir de una ventana de *MS-DOS* se teclea y ejecuta la siguiente instrucción:

C:\ti\carpeta de proyectos\proyecto\hex500 –m1 –boot –e 500h –romwidth 16 –memwidth 16 –map bootmap –bootorg 8000h archivo.out –o archivo.hex

Donde:

Carpeta de proyectos: ubicación del archivo.
Proyecto: carpeta donde se encuentra el proyecto.
Archivo.out: archivo de salida en formato COFF de la aplicación.
Archivo.hex: archivo que generará la utileria hex500.

- **Programar memoria flash.** Desde la ventana de MS-DOS se teclea y ejecuta:

C:\ti\c5400\dsk\utilities\flashu 2 –l c:\ti\carpeta de proyectos\proyecto\archivo.hex. El CCS no debe estar abierto durante la grabacion de la memoria flash. [2]

Interruptor	Estado.
1	Encendido (abajo)
2	Encendido (abajo)
3	Encendido (abajo)
4	Encendido (abajo)
5	Apagado (arriba)
6	Apagado (arriba)
7	Encendido (abajo)
8	Encendido (abajo)

Tabla 4.1 Posiciones de los Interruptores (Port1) para la Grabación de la Memoria Flash.

Una vez realizado el procedimiento anterior se apago la tarjeta, se removió el jumper localizado en los pines 1 y 2 en **JP2**, se encendió nuevamente la tarjeta y se presionó el botón de reset, y se obtuvo un funcionamiento del sistema independiente de la PC.

4.5 Resumen.

En este capítulo se presentó la implementación del Sistema de Efectos de Audio en Tiempo Real desde los primeros intentos de programas en el DSP hasta su grabación en la memoria flash de la tarjeta de desarrollo.
La implementación del sistema se realizó por partes, teniendo funcionando los efectos especiales por separado para después unirlos en un solo proyecto.

La rutina de selección de los efectos es muy sencilla y aprovecha los dos interruptores de usuario que tiene la tarjeta de desarrollo.
Una vez que el sistema se implementó, depuró y se grabó en la memoria flash de la tarjeta de desarrollo, teniendo un sistema que funciona independientemente de la presencia de una PC.

Referencias.

[1] Escobar S. L., Psenicka B. y Molero A. *Arquitectura de DSPs, familias TMS320C54x y TMS320C54XX y aplicaciones.* Facultad de Ingeniería, UNAM, Enero 2005.

[2] Zaragoza. E.A. *TMS320VC5402 Flash Bootloader, Nota de Aplicación Inédita.* Revisión B. Facultad de Ingeniería UNAM. México 2006.

CAPÍTULO 5.
PRUEBAS Y RESULTADOS.

En este capítulo se muestran los resultados obtenidos a través de una serie de pruebas que se realizaron al sistema de efectos especiales de audio, incluyendo la simulación de cada uno de los efectos en Matlab.
Se presenta también una comparación entre el comportamiento de los efectos de audio simulados y los implementados en el DSP, tanto en el dominio del tiempo como en la frecuencia, mostrando los errores que existen entre ellos.
Debido a que para el manejo de las señales como la voz en tiempo real, no es tan fácil verificar los efectos probados, se utilizaron señales simples como senoidal y pulso para probar el sistema y los algoritmos. Posteriormente el sistema se probó en tiempo real con señales como voz y audio con los efectos descritos.

Por último se incluye una evaluación del desempeño del sistema a nivel de hardware, como son tiempos de proceso, memoria de datos y de programa.

5.1 Visualización de los Efectos Especiales de Audio en el DSP.

Una vez teniendo el sistema de efectos especiales de audio funcionando en tiempo real, se procedió a determinar su comportamiento a través de una serie de pruebas para observar el resultado de aplicar los efectos de especiales de audio sobre diferentes señales tanto en el dominio del tiempo como en el de la frecuencia.
Las señales utilizadas para efectos de pruebas fueron un pulso, una senoidal y una señal de audio.

Señal Pulso.
Se utilizó una señal pulso de 1024 puntos, de los cuales tienen valor de uno los primeros quince puntos y de cero los restantes.

Señal Seno.
Se utilizó una señal senoidal de 1024, teniendo dos ciclos completos.

Señal de Audio.
La señal de audio utilizada fue grabada a $f_S = 8000Hz$ y consta de 1024 puntos.

Una vez teniendo la señal de prueba y su espectro se procedió a aplicarle cada uno de los efectos de audio, para obtener la gráfica de la señal modificada así como el espectro resultante de dichas modificaciones. El espectro se obtuvo mediante la Transformada Rápida de Fourier (FFT) de 1024 puntos.

Con el fin de tener un parámetro para el análisis y comparación del comportamiento de los efectos de audio implementados en el DSP, se realizó la simulación de cada uno de los efectos en Matlab, ajustando los parámetros de cada efecto al mismo valor que tienen en el sistema realizado en el DSP,

Contando con los comportamientos de los efectos de audio tanto en la simulación como en su implementación en el DSP se realizó una comparación para determinar si es que existen o no diferencias entre dichos comportamientos y en caso de existir determinar de que magnitud son tales diferencias.
Los resultados obtenidos en el DSP se guardaron en un archivo "*.dat*" con el fin de poder manejar esos datos en Matlab y así obtener tanto graficas comparativas como los errores existentes entre la simulación y la implementación en el DSP.

Se obtuvieron alrededor de 80 gráficas, entre las generadas por el sistema implementado en el DSP y las generadas por la simulación de los efectos en Matlab, así como las gráficas comparativas entre la simulación y al implementación en el DSP, observando que las graficas obtenidas en el DSP para cada efecto y cada señal son muy similares a sus respectivas simulaciones.

5.1.1 Efectos de Audio en el Dominio del Tiempo.

Se muestran a continuación las gráficas resultantes más importantes y que más datos aportan sobre el comportamiento del sistema.

Eco.
Para este efecto de audio la gráfica mas representativa es la obtenida de aplicar el efecto Eco a una señal pulso, el resultado se presenta en la *figura 5.1*.

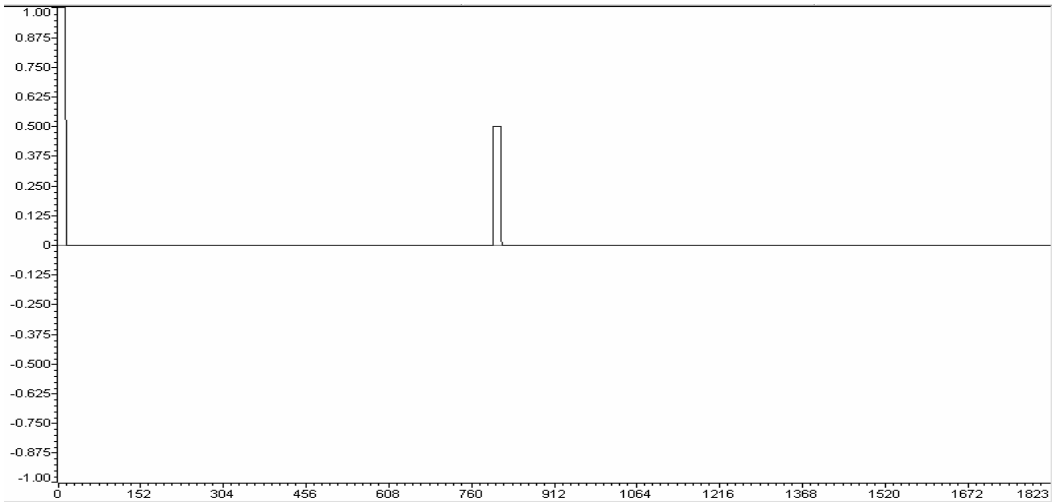

Figura 5.1 Señal Pulso con Eco en el DSP.

Se observa en la *figura 5.1* la definición básica del efecto Eco, que es la repetición de la señal original después de retraso, en este caso de 800 muestras, y con una atenuación notable del 50%, es evidente también que la señal resultante es de mayor tamaño que la original siendo de 1824 puntos, y este aumento es igual al valor del retraso.

Reverb.
Al igual que en el efecto Eco, la gráfica mas representativa para este efecto es la resultante de aplicar el efecto Reverb a una señal pulso, como se muestra en la *figura 5.2.*

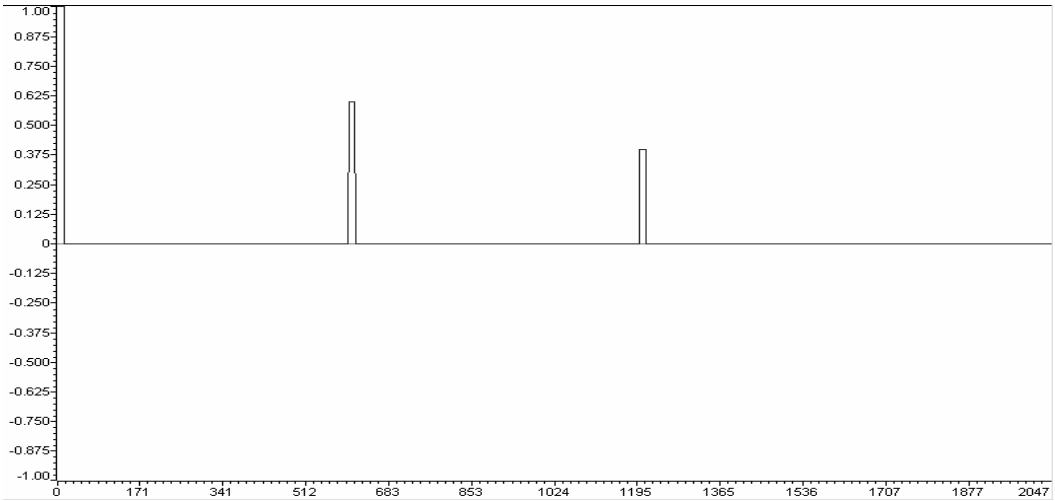

Figura 5.2 Señal Pulso con Reverb en el DSP.

Se observa en la *figura 5.2* que el comportamiento de este efecto es el descrito en el *Capítulo 2*, teniendo dos repeticiones de la señal original con retrasos de 600 y 1200 muestras, con sus respectivas atenuaciones, 60% y 40%, siendo la repetición final la más atenuada, al igual que en el efecto Eco se observa que la señal resultante es de mayor tamaño que la original, en este caso la señal resultante consiste en 2224 puntos, el incremento corresponde al valor del retraso mayor.

Flanger.

Para este efecto de audio la prueba se realizó con un tren de pulsos, teniendo un pulso de 15 puntos cada 100 muestras, y el resultado de aplicar el efecto Flanger a esta señal se muestra en la *figura 5.3.*

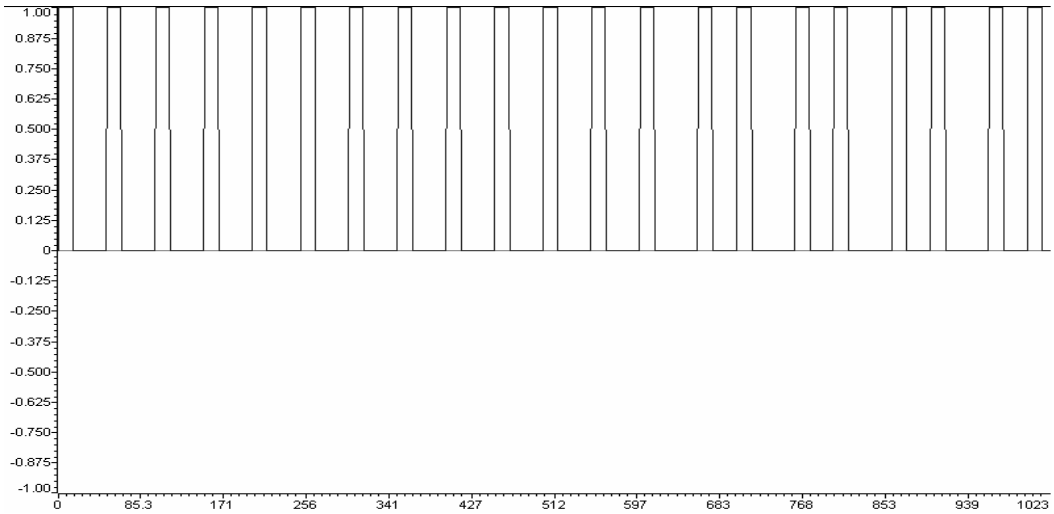

Figura 5.3 Señal Tren de Pulsos con Flanger en el DSP.

Al analizar la *figura 5.3* se observa varias repeticiones de la señal, lo que provoca cambios en la forma de la señal resultante, también se observa que dichas repeticiones se pueden presentar en cualquier parte de la señal y que los retrasos son mas pequeños que para el efecto Eco y el efecto Reverb, siendo de un máximo de 100 muestras, también es visible que la señal resultante es del mismo tamaño que la señal original, es decir, 1024 puntos.

Silenciar.

La gráfica mas representativa para este efecto es la resultante de generar un efecto Silenciar a una señal Senoidal como se muestra en la *figura 5.4*, sin embargo para efectos de prueba se cambió la frecuencia de la señal de ganancia para que el efecto sea más notorio, teniendo una $f = 80\,Hz$ en lugar de una $f = 5.333\,Hz$, ya que la frecuencia implementada en el DSP es muy pequeña para utilizarla en una señal de 1024 puntos.

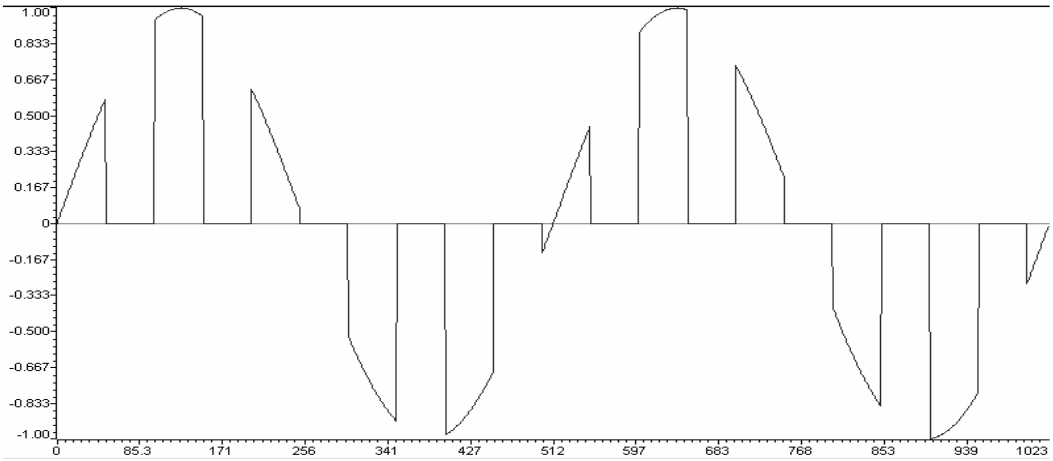

Figura 5.4 Señal Senoidal Silenciada en el DSP.

En la *figura 5.4* se observa el comportamiento del efecto Silenciar, que es básicamente hacer cero cierto número de puntos de la señal, se observa también que la señal resultante es del mismo tamaño que la original, es decir, 1024 puntos.

5.1.2 Evaluación de Errores en el Dominio del Tiempo.

Se realizó la comparación entre las simulaciones y la implementación en el DSP para cada efecto especial de audio, utilizando las tres señales mencionadas anteriormente, se evaluó el error cuadrático medio con ayuda de Matlab, siguiendo la siguiente ecuación [1]:

$$Ecm = \frac{\sum_{n=1}^{N} \left| X_{MAT}(n) - X_{DSP}(n) \right|^2}{N} \tag{5.1}$$

Donde:

Ecm = *Error Cuadrático Medio.*

$X_{MAT}(n)$ = *Señal Simulada en Matlab.*

$X_{DSP}(n)$ = *Señal Obtenida en el DSP.*

N = *Número de Muestras de la Señal* (1024).

Los errores cuadráticos medios obtenidos se muestran en la *tabla 5.1*.

Señal	**Efecto**			
	Eco	*Reverb*	*Flanger*	*Silenciar*
Pulso	12.718×10^{-6}	3.1475×10^{-18}	0.97656×10^{-3}	0
Senoidal	0.10399×10^{-9}	0.11606×10^{-9}	2.8×10^{-3}	1.9×10^{-3}
Audio	31.507×10^{-6}	1.8961×10^{-6}	19.9×10^{-3}	1×10^{-3}

Tabla 5.1 Error Cuadrático Medio en el Dominio del Tiempo.

En la *tabla 5.1* se observa que el error cuadrático medio es casi despreciable y que el valor más grande se obtiene cuando se produce un efecto Flanger a una señal de Audio. La *figura 5.5* muestra la comparación entre las señales de Audio con Flanger generadas por la implementación en el DSP y por la simulación en Matlab.

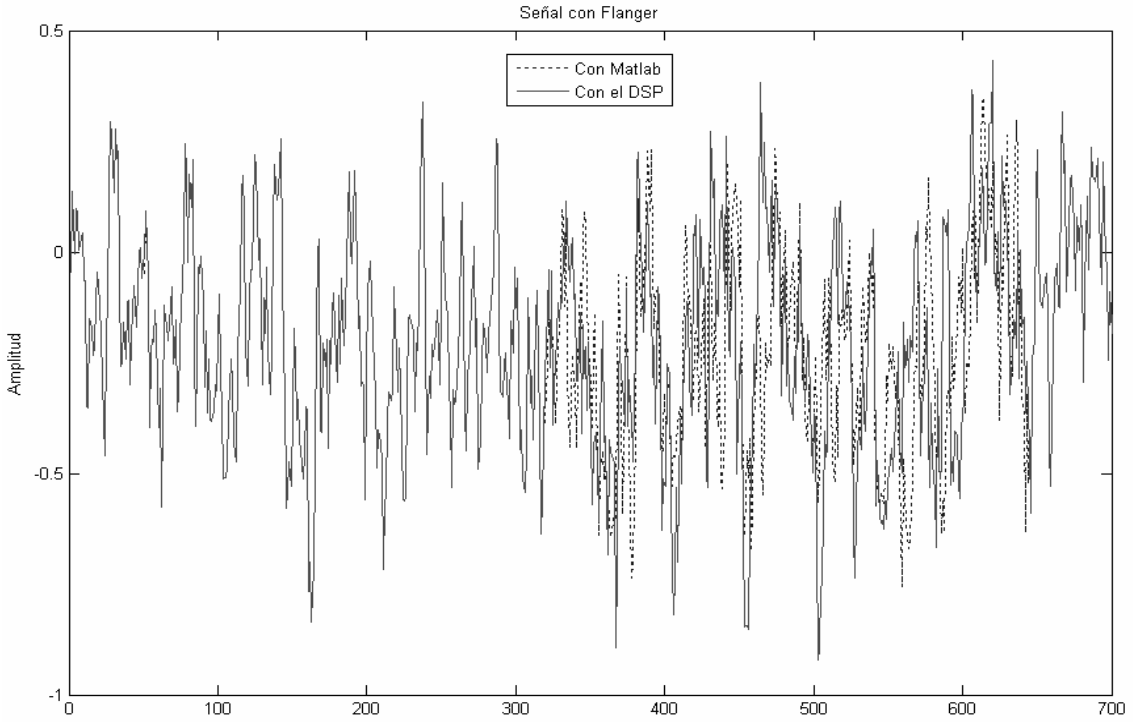

Figura 5.5 Comparación de la Señal de Audio con Flanger.

En la *figura 5.5* se observa que los errores se producen solo en ciertas partes y no a lo largo de toda la señal, estos puntos corresponden a los puntos donde se aplica la suma o la repetición de la señal, producto del efecto Flanger, el cual difiere en sus dos implementaciones debido a que el método para producir este efecto consiste en variar el retraso de la señal que se suma a la original mediante el uso de una señal senoidal, en el caso de Matlab es muy fácil de implementar ya que solo se necesita un ciclo "*for"* para hacer el barrido de la señal, y dentro de dicho ciclo se puede generar la onda senoidal basado en el valor del número de muestra que la variable que sirve para barrer la señal presente en cada instante, sin embargo esto no se puede implementar en el sistema de efectos de audio debido a que debe funcionar en tiempo real, por lo que la señal senoidal se va generando por una función diseñada para tal fin, dicha función genera una onda senoidal mediante el uso de una variable que va cambiando de valor cada vez que la función es llamada, al realizar dos ciclos completos de la onda senoidal la variable utilizada regresa a su valor inicial y comienza de nuevo el ciclo. Debido a lo anterior se produce una diferencia en la muestra que se debe sumar a la señal original en su implementación en el DSP, sin embargo esto no afecta el desempeño del efecto Flanger ya que dicha diferencia entre muestras es muy pequeña, entre 1 y 2 muestras, y cuando se produce este efecto en una señal de audio en tiempo real no es notorio el error.

5.1.3 Efectos de Audio en el Dominio de la Frecuencia.

El análisis del comportamiento de los efectos de audio en el dominio de la frecuencia es de gran ayuda para evaluar el desempeño general del sistema de efectos especiales de audio que se implementó. En las *figuras 5.6* a *5.10* se presentan las gráficas mas importantes de este análisis para cada efecto especial de audio.

Eco.

Al igual que el análisis en el dominio del tiempo, la gràfica que más información aporta es la del espectro que resulta de generar un efecto de Eco a una señal Pulso, como se muestra en la *figura 5.6.*

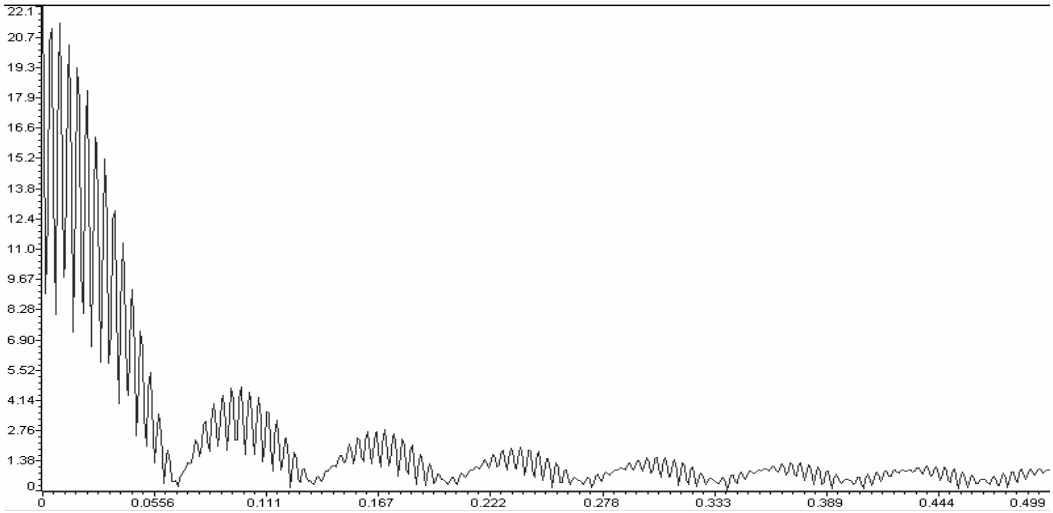

Figura 5.6 Espectro de la Señal Pulso con Eco en el DSP.

La *figura 5.6* nos muestra como afecta el efecto Eco al espectro a la señal Pulso en el dominio de la frecuencia, lo más evidente e importante de mencionar es un solapamiento, es decir que se esta encimando el espectro debido a que se tiene una repetición de la señal original.

Reverb.

Para este efecto nuevamente se tiene que la gráfica que presenta mayor solapamiento es la del espectro resultante de aplicar el efecto Reverb a una señal Pulso, como se muestra en la *figura 5.7.*

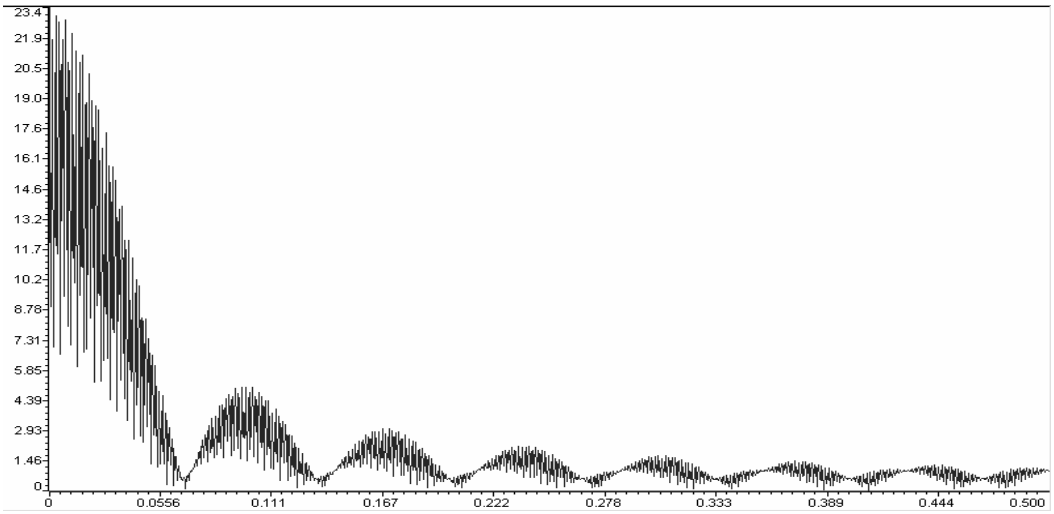

Figura 5.7 Espectro de la Señal Pulso con Reverb en el DSP.

El análisis de la *figura 5.7* resulta, como en el efecto Eco, en la identificación de un solapamiento del espectro, pero en mayor cantidad debido a la presencia de dos repeticiones de la señal original.

Flanger.

Para el efecto Flanger la gráfica con mayor información y la más relevante es la del espectro producido al aplicar este efecto a una señal Pulso.

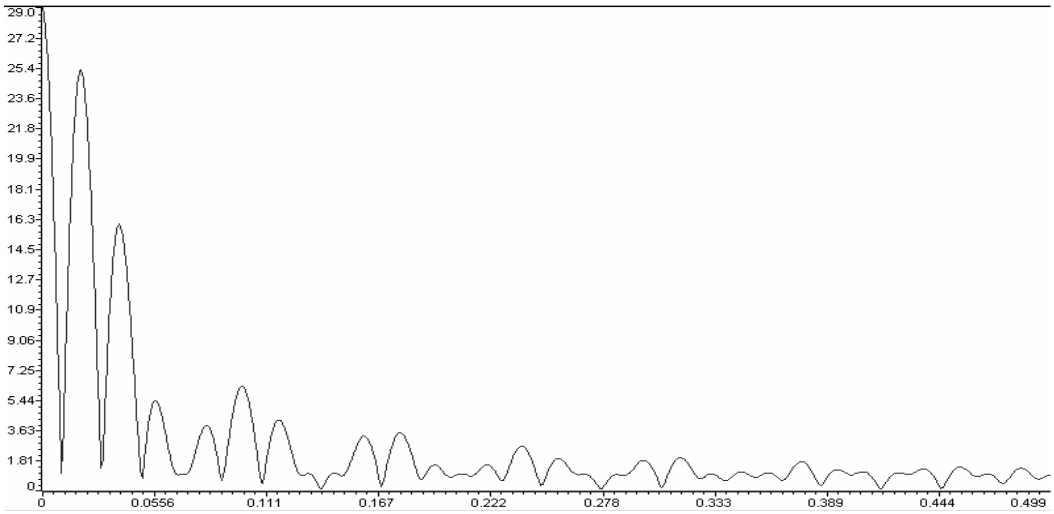

Figura 5.8 Espectro de la Señal Pulso con Flanger en el DSP.

La *figura 5.8* indica la presencia de componentes armónicas en el espectro de una señal con Flanger, lo que implica un cambio notable en el mensaje contenido en la señal, dicho cambio es precisamente lo que se pretende al implementar este efecto. Al escuchar una señal de voz con un efecto de Flanger se tiene que la voz cambia, es decir no se escucha igual que la voz original, y esto es debido a las armónicas observadas en el espectro.

Silenciar.

El espectro de la señal Senoidal con este efecto aporta más información al análisis, una vez mas se utilizo una $f = 80\,Hz$ para la prueba, como se muestra en la *figura 5.9.*

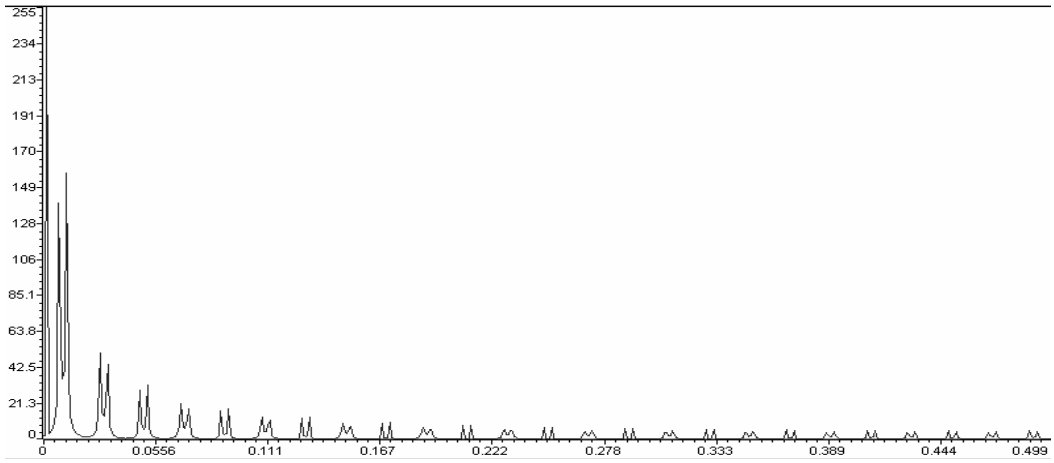

Figura 5.9 Espectro de la Señal Senoidal Silenciada en el DSP.

En la *figura 5.9* se puede observar la presencia de componentes armónicas, lo cual es predecible ya que el efecto silenciar cambia de manera muy notoria la forma de la señal en el dominio del tiempo y al ser otra señal diferente a una senoidal se presentan estas armónicas que son las componentes de la serie de Fourier que describe la forma de la señal resultante. Cabe mencionar que si la frecuencia de la señal de ganancia fuera más grande, la presencia de armónicas aumentaría en proporción a dicha frecuencia, además de que serían de armónicas de frecuencia cada vez más grande.

5.1.4 Evaluación de Errores en el Dominio de la Frecuencia.

Al comparar los espectros de las señales resultantes de aplicar los efectos especiales a las tres señales de prueba se observaron algunas diferencias entre las simulaciones y los resultados entregados pro el DSP, por lo que también se calculó el error cuadrático medio, basado en la *ecuación* (*5.1*), el error en el dominio de la frecuencia se calculó con [1]:

$$Ecm = \frac{\sum_{n=1}^{N} \left| Y_{MAT}(n) - Y_{DSP}(n) \right|^2}{N} \tag{5.2}$$

Donde:

$Ecm = Error\ Cuadrático\ Medio.$

$Y_{MAT}(n) = Espectro\ de\ la\ Señal\ Simulada\ en\ Matlab.$

$Y_{DSP}(n) = Espectro\ de\ la\ Señal\ Obtenida\ en\ el\ DSP.$

$N = Número\ de\ Muestras\ del\ Espectro\ de\ la\ Señal.$ (1024)

En la *tabla 5.2* se muestra el error cuadrático medio de todos los espectros obtenidos.

Señal	**Efecto**			
	Eco	*Reverb*	*Flanger*	*Silenciar*
Pulso	11.4×10^{-3}	2.8×10^{-15}	486×10^{-3}	0
Senoidal	26.251×10^{-9}	37.454×10^{-9}	1.0334	423.9×10^{-3}
Audio	30×10^{-3}	1.7×10^{-3}	7.9290	498.7×10^{-3}

Tabla 5.2 Error Cuadrático Medio en el Dominio de la Frecuencia.

En la *tabla 5.2* se observa que el error cuadrático medio es casi despreciable y el valor más grande se obtiene cuando se produce un efecto Flanger a una señal de Audio. La *figura 5.10* muestra la comparación entre los espectros de las señales de Audio con Flanger generadas por la implementación en el DSP y por la simulación en Matlab.

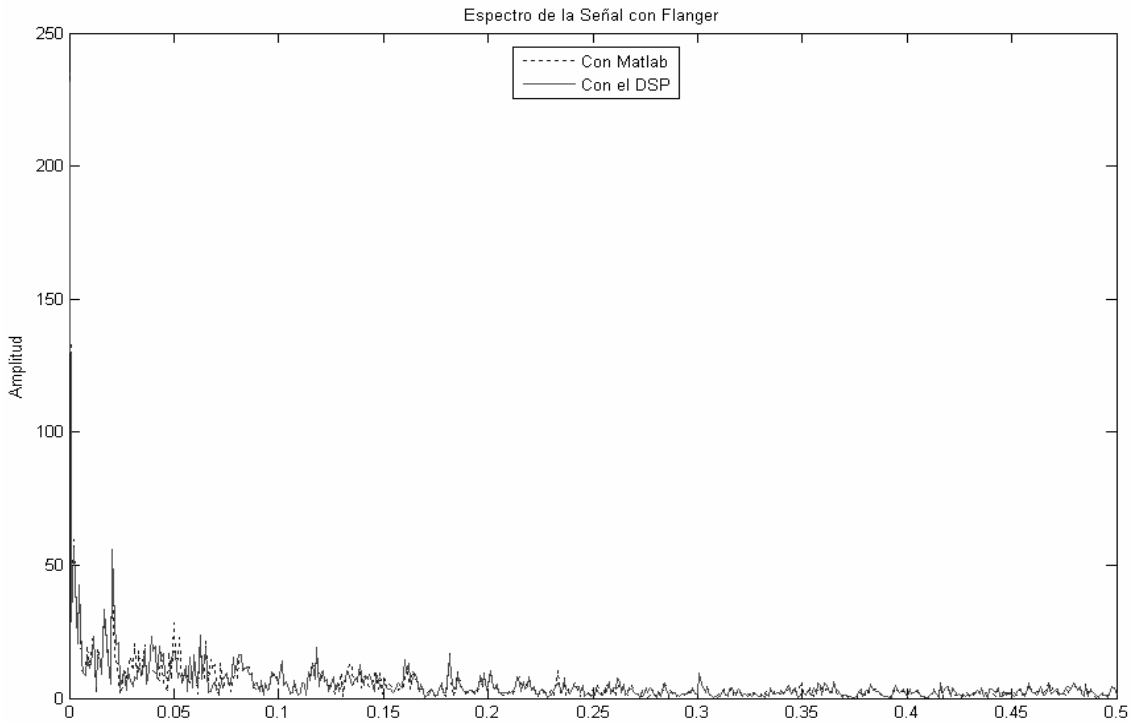

Figura 5.10 Comparación del Espectro de la Señal de Audio con el Efecto Flanger.

Las diferencias que se presentan entre los espectros mostrados en la *figura 5.10* son provocadas por la implementación en tiempo real del efecto Flanger, sobre todo en las diferencias en la generación del retraso por medio de una onda senoidal como se mencionó en el análisis del error en el dominio del tiempo.

5.2 Análisis de los Recursos del Sistema.

Una parte importante del análisis del funcionamiento del sistema es determinar cuanta capacidad del DSP se usa, dicha capacidad se refiere a memoria dato, memoria programa, puertos y los tiempos de ejecución.

Memoria Dato.

La memoria dato definida por el archivo de comandos (*.cmd*) (ver *Anexo 1*), esta definida a partir de la localidad $0x1F00$ y con una longitud de $0x3F7F$, es decir, 16255 localidades de memoria, sin embargo el sistema implementado no ocupa en su totalidad dichas localidades de memoria. El análisis resulta en que el sistema implementado utiliza una memoria dato de 2557 localidades, comprendidas entre las localidades $0x1F00$ y $0x28FD$.

Memoria Programa.

El archivo de comandos define la memoria programa a partir de la localidad $0x0100$ y con una longitud de $0x3F70$ siendo de un total de 16240 localidades de memoria. El sistema implementado utiliza la memoria programa comprendida entre las localidades $0x0100$ y $0x22B7$, es decir, 8631 localidades de memoria programa.

Evaluación de Tiempos de Ejecución.

Los tiempos de ejecución se refieren, al tiempo que tarda cada efecto en realizarse dentro de la interrupción principal del programa. El sistema implementado está diseñado para ser utilizado principalmente para señales de voz por lo que la frecuencia de muestreo se estableció en $f_S = 8000\ Hz$, por lo que la interrupción se produce cada que una muestra nueva de la señal es adquirida por el DSP, esto es, cada $125\ \mu seg$. Para obtener los tiempos de ejecución se utilizó la opción "*profiler*" incluida dentro del Code Composer Studio, la cual permite obtener los ciclos de instrucción requeridos para la ejecución de una parte o la totalidad de un programa. El tiempo de ejecución se obtiene a partir de los ciclos de instrucción, sabiendo que la frecuencia del cpu del DSP que se esta utilizando es de $100\ MHz$, cada ciclo de instrucción requiere un tiempo de 10 *nseg* para completarse.

La *tabla 5.3* muestra los resultados obtenidos referentes a ciclos de instrucción y tiempo de ejecución para cada efecto especial implementado.

Efecto.	**Ciclos de Instrucción.**	**Tiempo de Ejecución.** $[\mu seg]$
Eco.	3041	30.41
Reverb.	4168	41.68
Flanger.	3300	33.00
Silenciar.	3017	30.17

Tabla 5.3 Ciclos de Instrucción y Tiempos de Ejecución.

Como se observa en la *tabla 5.3* el efecto que más tiempo de ejecución requiere es el Reverb, por lo que al estar realizándose este efecto, es cuando más trabaja el DSP, utilizando un 33.344% del tiempo entre muestra y muestra (*figura 5.11*), el resultado anterior es comprensible debido a que el efecto Reverb requiere más cálculos para su funcionamiento, debido a que como se muestra en el *Capítulo 4*, se deben mover las muestras de la señal de entrada almacenadas en el buffer de memoria, que para este efecto es de 1200 localidades.

La *figura 5.11* muestra una comparación entre el tiempo de interrupción y el máximo tiempo consumido por el DSP.

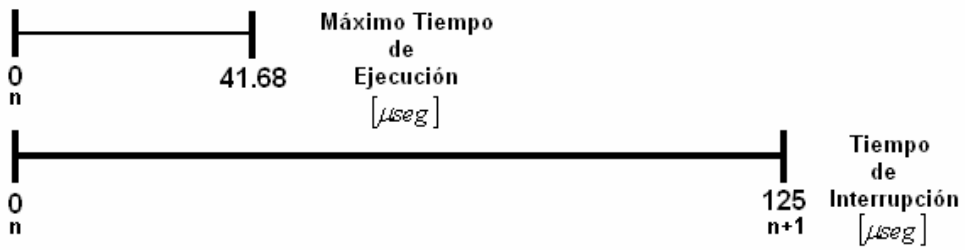

Figura 5.11 Tiempo entre Muestra y Muestra y Tiempo de Ejecución del Efecto Reverb.

Al analizar la *figura 5.11* se observa que después de realizado el efecto el sistema aun dispone de tiempo para realizar otros procesos más complicados antes de se presente de nuevo la interrupción, hablando en porcentaje se dispone de un 66.656% del tiempo total.

Puertos Utilizados.

En este sentido el sistema ocupa muy pocos puertos, solo se encuentran en uso la interfaz de micrófono y altavoz (McBSP1) y el puerto paralelo 1 (Port1). Es decir el sistema todavía dispone de muchos recursos para realizar otras aplicaciones.

5.4 Resumen.

En este capítulo se presentaron las pruebas realizadas a los efectos de audio por medio de las gráficas más representativas para cada uno de los efectos especiales de audio implementados en el sistema, tanto en el dominio del tiempo como en el de la frecuencia.
Se mostró los errores que se tienen al comparar las simulaciones hechas en Matlab y los resultados obtenidos en el DSP, incluyendo la gráfica del efecto en donde se presenta el mayor error, así como la del espectro que igualmente presenta el mayor error.

En cuanto a los recursos del sistema se mostraron las memorias dato y programa, así como los tiempos de ejecución requeridos por cada efecto especial.

Referencias.

[1] George C. Canavos, *Probabilidad y Estadística, Aplicaciones y Métodos*, Mc Graw Hill, primera edición, México 1988.

CAPÍTULO 6.
CONCLUSIONES.

Se ha realizado la implementación de cuatro efectos especiales de audio en tiempo real utilizando una tarjeta con un DSP.

El sistema de efectos especiales de audio en tiempo real implementado en el DSP tiene varias aplicaciones posibles: se puede utilizar para modificar la voz en la grabación de piezas musicales, para enfatizar y crear efectos especiales de audio para películas y obras de teatro, e incluso utilizarlo con algún instrumento musical al cual se le desee modificar su sonido habitual, como por ejemplo una guitarra eléctrica o un bajo.

Todas las aplicaciones posibles mencionadas en la implementación final de este trabajo son independientes de la presencia de una PC, es decir, se ejecutan en la tarjeta del DSP, limitando sus necesidades a solo una toma de corriente. Los elementos de entrada pueden ser un micrófono o un instrumento musical y los elementos de salida son principalmente bocinas y/o amplificadores de audio.

En cuanto al desempeño auditivo del sistema se tiene un funcionamiento en tiempo real aceptable, percibiendo al encender el sistema un transitorio en cada efecto especial, manifestado como ruido, debido a que en un principio la memoria del DSP tiene valores aleatorios, pero una vez que se llenan los buffers de memoria para cada efecto el sistema tiene un buen funcionamiento. La selección de los efectos de audio es muy fácil ya que solo se debe cambiar la posición de los switches de usuario para cambiar de efecto.

El análisis del sistema en el dominio del tiempo y de la frecuencia arrojan datos que nos ayudan a entender mejor las características de su funcionamiento, principalmente para los efectos Eco y Reverb que presentan un ruido muy pequeño en su funcionamiento, encontrando que la causa del ruido es inherente a los efectos. El efecto Flanger se puede decir que distorsiona la voz que pasa por el sistema solo al escuchar el resultado de su aplicación, sin embargo, el estudio del espectro nos muestra cual es la causa de esa distorsión, que en este caso es la presencia de armónicas en el espectro. Dicho análisis también revela que la implementación final del sistema realiza los efectos de audio de manera correcta al no existir mucha diferencia con la simulación de los efectos de audio en Matlab y que el error cuadrático medio para todos los efectos es muy pequeño.

En cuanto a los recursos del sistema se tiene que la aplicación implementada consume una pequeña parte de la capacidad total de la tarjeta de desarrollo y del DSP, en cuanto a la memoria dato y programa, y que solo se utiliza un puerto de la tarjeta en este caso la interfaz de micrófono y altavoz.

Al obtener los tiempos de proceso se observa que sólo se utiliza el 33.344% del tiempo total disponible entre interrupciones, lo cual deja bastante tiempo para implementar otras aplicaciones en el sistema sin afectar su funcionamiento en tiempo real.

El sistema se puede expandir para tener una amplia diversidad de efectos de audio si se implementa un puerto selector externo con más switches y se agregan las rutinas necesarias al programa principal.

ANEXOS.

Anexo1. Códigos Fuente en Lenguaje C.

Sistema.c:

```
/* Sistema de Efectos Especiales de Audio en Tiempo Real

Agregar Librerías al Proyecto:

rts.lib
drv5402.lib
dsk5402.lib
type.h
board.h
codec.h */

#include <type.h>
#include <board.h>
#include <codec.h>                                    //Incluye archivos .h
#include <variables.h>
#include <funciones.h>
#include <math.h>

#define GLOBAL_INT_ENABLE     asm( " rsbx intm ")  //Define habilitación
                                                      de interrupción.
#define GLOBAL_INT_DISABLE    asm( " ssbx intm ")//Define deshabilitación
                                                      de interrupción

volatile short *imr = (short *) 0x00;  //Define registro de interrupción
volatile short *ifr = (short *) 0x01;
volatile short *pmst= (short *) 0x1d;
volatile short *drr = (short *) 0x41; //Define registro de recepción
volatile short *dxr = (short *) 0x43; //Define registro de transmisión

HANDLE hHandset;

interrupt void recibir(void)  //Interrupción
    {
     short datoP1;  //Define ariable
     data=*drr;     //Recibe muestra
      datoP1 = port1;  //Lee switches de usuario
      switch(datoP1)  //Selección de efecto
      {
            case 0:
                  data = reverb(data);
                  break;
            case 32:
                  data = eco(data);
                  break;
            case 64:
                  data = flanger(data);
                  break;
            case 96:
                  data = silencio(data);
                  break;
      }

     *dxr=data; //Transmite resultado
    }
```

```
void main(void)  //Rutina principal
    {
      GLOBAL_INT_DISABLE;                    //Deshabilita interrupciones

      brd_init(100);                         //Frecuencia de CPU 100 MHz

      hHandset = codec_open(HANDSET_CODEC);              //Inicia codec

      codec_dac_mode(hHandset, CODEC_DAC_15BIT);   //Configura conversión
      codec_adc_mode(hHandset, CODEC_ADC_15BIT);     analógica digital y
      codec_ain_gain(hHandset, CODEC_AIN_0dB);       digital analogica
      codec_aout_gain(hHandset, CODEC_AOUT_MINUS_6dB);
      codec_sample_rate(hHandset,SR_8000);         //fs=8000 Hz

      *pmst = 0x00A0;                           //Vectores de interrupción
      *imr |= 0x0400;         //Habilita interrupción de recepción McBPS1
      *ifr = *ifr;

      GLOBAL_INT_ENABLE;                     //Habilita interrupciones

      while(1){}                             //Loop infinito
    }
```

Variables.h:

```
/*Variables a utilizar*/

#define N 800
#define Nu 100
#define Nr 600
#define Ns 1500
#define pi 3.1416
#define f 0.5                        //Definición de variables a utilizar
#define G 1
#define fs 8000

short data,n,retraso,c,g,fc;
int buff[N+2];
int buff2[Nu+2];
int buff3[Nr*2+2];
```

Funciones.h:

```
/*****************Eco*****************************/

/*Funcion que realiza el Eco*/

Short eco(short nvalor)
    {
      int i;                                //Define
      short res=0;                            variables
      buff[N+1]=nvalor;                 //Coloca muestra actual en el buffer
      res=data+0.5*buff[1];           //Suma la muestra actual y la retrasada
      for (i=N; i>-1; i--)
          {
            buff[N-i]=buff[N-i+1];        //Recorre muestras en el buffer
          }
      return (short) res;                 //Regresa resultado
    }

/*****************Flanger******************/

/*Funcion para redondear*/

short round(short valor)
    {
      int j;
      short b=0;                                        //Define
      short rou=0;                                        variables
      for (j=0; j<10; j++)
      {
      b=valor-j;
      if (b<0.5)
       { rou=j;
         break;}                                        //Compara valor
       if (b<1)
       { rou=j+1;
         break;}
      }
      return (short) rou;                               //Regresa resultado
    }

/*Funcion varia el retraso*/

short delay(void)
   {
      short m=0;                                      //Define variable
      if (n==16000)
      {
       n=0;
      }
      m=(1+4*(1+sin(2*pi*f*(n++)/8000)));             //Genera retraso
      m=round(m);                                  //Llama función de redondeo
      return (short) m;                            //Regresa resultado
    }
```

```
/*Funcion que realiza el Flanger*/

short flanger(short nvalor)
    {
      int i;                                                    //Define
      short res=0;                                                variable
      retraso=delay();                          //Llama función de retraso
      buff2[Nu+1]=nvalor;            //Coloca muestra actual en el buffer
      res=data+G*buff2[1+(retraso*(Nu/10))];    //Suma muestra actual y
                                                  la retrasada
      for (i=Nu; i>-1; i--)
          {
          buff2[Nu-i]=buff2[Nu-i+1];      //Recorre muestras en el buffer
          }
      return (short) res;                 //Regresa resultado
   }

/*****************Reverb*****************/

short reverb(short nvalor)
    {
      int i;                                      //Define
      short res=0;                                  variables
      buff3[Nr*2+1]=nvalor;          //Coloca muestra actual en el buffer
      res=data+0.6*buff3[Nr+1]+0.4*buff3[1];     //Suma muestra actual y
                                                    las retrasadas
      for (i=Nr*2; i>-1; i--)
          {
          buff3[Nr*2-i]=buff3[Nr*2-i+1];  //Recorre muestras en el buffer
          }
      return (short) res;                 //Regresa resultado
    }

/****************Silencio******************/

 short silencio(short nvalor)
    {
      short res=0;                        //Define variable
      fc=(fs/Ns);                         //Calcula frecuencia de silencio
      if(n==Ns)
      {
      n=0;
      }
      c=sin(2*pi*fc*(n++)/8000);          //Señal base
      if (c>0)
      {
       g=2;                               //Ganancia 2
      }
      else
      {
       g=0;                               //Ganancia 0
      }
      res=g*nvalor;              //Multiplica muestra actual por ganancia
      return (short) res;                 //Regresa resultado
    }
```

Vectors54xx.asm:

```
/* Archivo que define interrupción por recepción de datos*/

    .mmregs

    .sect    " vectors"

      .ref   _c_int00
      .ref     _recibir

    .align  0x80

reset:  b _c_int00
        nop
        nop

nmi:  rete
            nop
            nop
            nop

sint17:     rete
            nop
            nop
            nop

sint18:     rete
            nop
            nop
            nop

sint19:     rete
            nop
            nop
            nop

sint20:     rete
            nop
            nop
            nop

sint21:     rete
            nop
            nop
            nop

sint22:     rete
            nop
            nop
            nop

sint23:     rete
            nop
            nop
            nop


sint24:     rete
```

```
            nop
            nop
            nop
sint25:     rete
            nop
            nop
            nop

sint26:     rete
            nop
            nop
            nop

sint27:     rete
            nop
            nop
            nop

sint28:     rete
            nop
            nop
            nop

sint29:     rete
            nop
            nop
            nop

sint30:     rete
            nop
            nop
            nop

int0:       rete
            nop
            nop
            nop

int1:       rete
            nop
            nop
            nop

int2:       rete
            nop
            nop
            nop

tint0:      rete
            nop
            nop
            nop

brint0:     rete
            nop
            nop
            nop

bxint0:     rete
            nop
```

```
            nop
            nop

dmac0:      rete
            nop
            nop
            nop

tint1:      rete
            nop
            nop
            nop

int3:       rete
            nop
            nop
            nop

hpint:      rete
            nop
            nop
            nop

brint1:     b _recibir
            nop
            nop

bxint1:     rete
            nop
            nop
            nop

dmac4:      rete
            nop
            nop
            nop

dmac5:      rete
            nop
            nop
            nop

            .end
```

Ligador.cmd:

```
/* Archivo ligador.cmd para el ligado de comandos */

MEMORY
   {
    PAGE 1: SCRATCH : origin = 0x0060, length = 0x0020
    STACK   : origin = 0x1180, length = 0x0560
    DATA   : origin = 0x1F00, length = 0x3F7F /*Memoria Dato*/

    PAGE 0: PROG : origin = 0x0100, length = 0x3F70 /*Memoria Programa*/
    PAGE 0: VECS : origin = 0x0080, length = 0x007f /*Vectores de
                                                      Interrupción*/
      }

SECTIONS
      {
      vectors > VECS          PAGE 0
      .text   > PROG          PAGE 0
      .cinit  > PROG          PAGE 0
      .switch > PROG          PAGE 0

      .const  > DATA          PAGE 1
      .data   > DATA          PAGE 1
      .bss    > DATA          PAGE 1
      .stack  > STACK         PAGE 1
      .trap   > SCRATCH       PAGE 1
      }
```